تحليل وتصميم الخوارزميات

Algorithms Design and Analysis

تأليف

الدكتور	الأستاذ المساعد	الباحث
حسن ياسين طعمه	هند رستم محمد شعبان	حسن ثابت رشيد كرماشة

hind_restem@yahoo.com
hassan_thabit@yahoo.com

تحليل وتصميم الخوارزميات

Algorithms Design and Analysis

الفهرس ....

الفصل الأول
مقدمة
(Introduction)

1-1: مقدمة في الخوارزميات (Algorithms Introduction) :

الخوارزمية هي مجموعه محددة من التعليمات (خطوات الحل) التي تؤدي إلى انجاز وظيفة (مهمة) معينة ويجب إن تتوافر فيها الشروط التالية :

1. المدخلات (Input) : صفر أو أكثر من القيم .
2. المخرجات (Output) : قيمة واحدة على الأقل .
3. الوضوح (Definiteness) : كل خطوة فيها (الخوارزمية) واضحة المعاني وغير غامضة أي يجب إن تفهم من قبل جميع الناس (علوم الحاسبات).

وعلى سبيل المثال نأخذ العبارة " Add 6 or 7 to X" هذه العبارة غير مسموح بها في الخوارزمية لأنها عبارة غير واضحة .

4. المحدودية (Finiteness) : كل خطوات الخوارزمية يمكن حلها في فترة زمنية محددة ، ولتوضيح ذلك نأخذ العبارة " قسم الرقم (10) على (3) بدقة عالية (كاملة)" هذه العبارة غير محدودة ويجب إن لا يسمح بها داخل البرنامج .
5. المحلولية (Effectiveness) : كل خطوة تكون ممكنة الحل أو الفعالية ، مثال ذلك العبارة " 0/3 " لا يمكن حلها ابداً .

يمكن لنا إن نوضح الفرق بين الخوارزمية والبرنامج حيث انه في النظرية الاحتسابية يوجد فرق بين الخوارزمية والبرنامج ، ففي الخوارزمية يجب إن تتوافر الشروط الخمسة الأنفة الذكر ويمكن وصفها بطرق عديدة مثل لغة طبيعية مع التأكيد على شرط الوضوح ، لغة خوارزمية (Pseudo code) ، مخططات انسيابية (Flow chart) ، بينما يمكن في البرنامج عدم تحقق الشرط الرابع حيث إن نظام التشغيل هنا هو الذي يعتمد على البرنامج ويوصف البرنامج بلغة الحاسبة حيث انه يصمم ليتحكم في تنفيذ مجموعه من الإعمال (Jobs) بحيث عند عدم توفر عمل معين فانه لا ينتهي من أعماله بل يستمر ويدخل في حالة انتظار لحين إدخال عمل جديد .

إن لكل لغة برمجية يوجد مترجم أو مفسر ولا يمكن تواجدهما معاً حيث إن المفسر يقوم بتنفيذ البرنامج خطوة خطوة (step by step) بينما المترجم فانه ينفذ البرنامج كاملاً ويظهر النتائج والأخطاء ،هذا يعني إن البرنامج هو عبارة عن خوارزمية وهيكل بياني أي انه طريقة لتنظيم البيانات.

خطوات تطوير البرنامج :

تمر عملية تطوير البرنامج بخمس خطوات رئيسية هي :

1. توصيف المتطلبات (Requirement specification):

هو تحديد المدخلات والمخرجات.

2. التصميم (Design):

هو تحديد العمليات الرئيسية التي تطبق على كل كيان بياني وافتراض وجود أجهزة معالجة لتنفيذ هذه العمليات.

3. التحليل (Analysis):

هو المفاضلة بين الخوارزميات المتوفرة التي تحل نفس المسألة تبعاً لمقاييس مفاضلة متفقاً عليها (تعقيدات الوقت ،تعقيدات الفراغ (الخزن)) باختيار أفضلها .

4. التحسين والتشفير (Refinement & Coding):

في هذه الخطوة يتم تحديد التمثيل البياني لكل كيان ثم كتابة الإجراءات لكل عملية على تلك الكيانات وتكوين نسخة متكاملة للبرنامج.

ملاحظة // التحليل يصلح الأخطاء اعتماداً على تعقيدات الخزن والوقت بينما التحسين يُصلح الأخطاء اعتماداً على النتائج الظاهرة في نهاية البرنامج.

5. التحقق من الصلاحية (Verification):

تتضمن هذه الخطوة ثلاث جوانب هي :

أ- البرهنة على الصحة (Proving) :

قبل استخدام البرنامج يجب إثبات انه صحيح حيث يتم استخدام الطرق المعروفة للبرهنة على الصحة .

ب- الاختبار (Testing):

هي عملية توليد نماذج بيانية يعمل عليها البرنامج حيث إن الهدف منها هو إعطاء إشارة على وجود أخطاء في البرنامج.

ج- تشخيص الأخطاء (Debugging):

عملية تحديد مواقع الأخطاء البرمجية في البرنامج وتصحيحها .

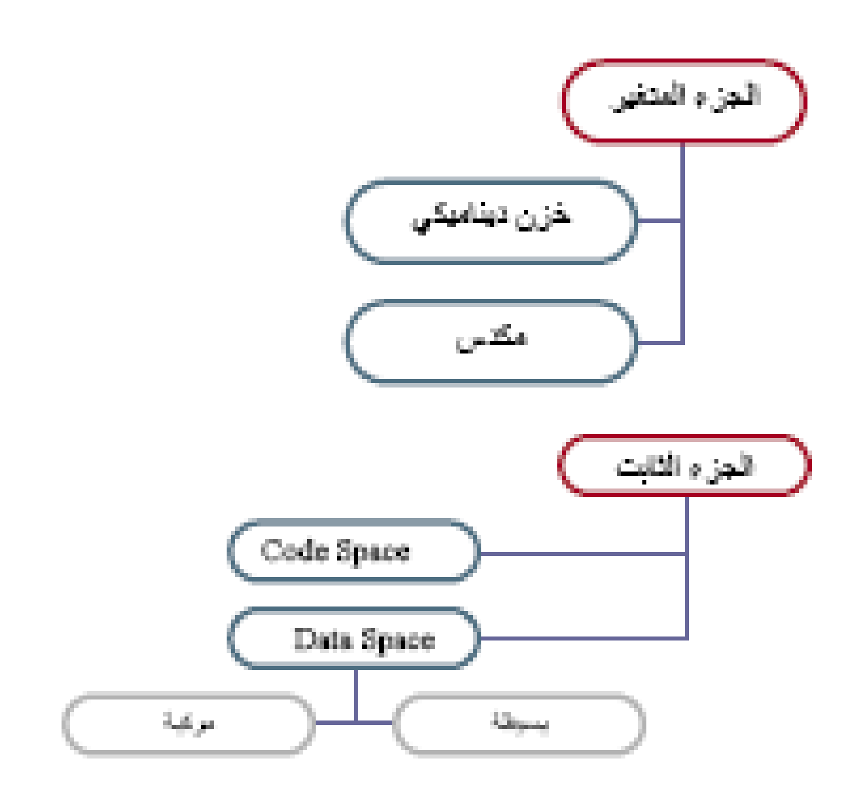
ملاحظة : إن التعريف يختلف عن الصلاحية فالتعريف هو معرفة شيء قد يكون صحيح أو خطأ بينما الصلاحية هي معرفة شي ء يجب إن يكون صحيح .
أما النموذج فهو تحقيق تمثيل بياني بالشكل الصحيح.
في علم الحاسبات يتم أولا الاختبار وبعدها يتم البرهنة بينما في علم الرياضيات يتم البرهان وبعده الاختبار.

2-1: كيفية تحليل الخوارزمية (Algorithm Analysis)

تحليل الخوارزمية هو تحديد الكفاءة للخوارزمية ومن ثم تحسينها حيث يوجد مقياسين مرتبطين مباشرة بانجازيه الخوارزمية هما :

1 - مقياس تعقيدات الفراغ أو الخزن (Space Complexity): هي كمية الذاكرة التي يتطلبها تشغيل البرنامج حتى اكتماله ،حيث يعتمد هذا النوع على جزئيين :

أ- جزء ثابت : أو مستقل عن خصائص المدخلات والمخرجات حيث يتضمن هذا الجزء فراغ التعليمات (Code space) ،الفراغ المخصص للمتغيرات (Data Space) سواء كانت البسيطة أو المتغيرات المركبة ذات الحجم الثابت إضافة إلى فراغ الثوابت ، ...... الخ .
ب- جزء متغير: يتألف من الفراغ الذي يتطلبه البرنامج بالمتغيرات المركبة والتي يعتمد حجمها على مثال المسالة المراد حله ، إضافة إلى فراغ المكدس المستخدم في التداخل (Reaction).



شكل(1): تحليل الخوارزمية

إن الخزن الديناميكي يمكن توضيحه بالمتغيرات التي يدخلها المبرمج (أي يدخل قيمها) وكذلك يتحكم بقِيمها فهي معتمدة على إدخال المبرمج للبرنامج.

أما القيم المركبة فهي المصفوفة التي تمثل بالمكدس حيث المؤشر هو (Sp) معمارياً وبرمجياً يسمى (Top) ، وفيما يخص تصميم الخوارزميات فان المهم هنا هو محتوى المصفوفة أي المكدس.

يمكن صياغة تعقيدات الخزن للبرنامج كالتالي :

ثابت

Code segment
Data segment
Heap segment
Stack segment

إن جزء (Code Segment) يمكن تمثله كما الدوال الجاهزة ، أما (Heap Segment) فيكون للمتغيرات التي يستخدمها البرنامج .

وعلية يمكن صياغة تعقيدات الفراغ S(p) للبرنامج (p)

$$S(p) = Const + Sp$$

حيث إن :

Const: تمثل جزء (Code segment) والمتغيرات البسيطة .

Sp : تمثل خصائص المثال .

2- تعقيدات الوقت (Time Complexity) : هي كمية الوقت التي يتطلبها تشكيل البرنامج حتى اكتماله ويتألف من :

$$T(p) = Const + tp$$

حيث :

Const: يمثل ثابت خاص بوقت الترجمة أو التأليف.

Tp: يمثل وقت تشغيل البرنامج .

مثال1// لبيان تعقيدات الفراغ (الخزن) والوقت لدالة معينة (بلغة ++C):

```
Float abc(float a,float b,float c)
{return(a+b+6*c+(a+b+c)/(a+b)+4.0); }
```

تعقيدات الفراغ أو الخزن :

تتطلب الدالة (abc) خمسة خلايا خزنية لخزن قيم المتغيرات (a,b,c) والمتغير الذي يحمل اسم الدالة وعنوان العودة (Return address) وهو خزن ثابت لا يعتمد على خصائص المثال(a,b,c).

$$S_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا تعني إن الخزن ثابت أي لا يعتمد على خصائص المثال .

تعقيدات الوقت : تستخدم صيغة عد الخطوات (Steps Count) لقياس تقدير الوقت حيث إن عدد الخطوات لهذه الدالة يساوي واحد بولهذا فان :

$$T_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا أيضا تعني إن الوقت ثابت.

مثال2// اكتب خوارزمية لإيجاد القاسم المشترك الأعظم (Greatest Common Divisor) GCD لعددين صحيحين .

الحل//

Step 0 : [check m and n]
   If m <= 0 or n <= 0 then
      Print error.
Step 1: [test m and n]
   If m < n then
      Inter change m by n
Step 2: [find the reminder]
   Divid m by n and let r is
   Reminder we will have 0 <= r < n
Step 3: [is r = zero]
   If r = 0 then
    GCD = n and exit.
Step 4:[inter change]
   M $\leftarrow$ n, n $\leftarrow$ r go to step 2

مثال تطبيق 1//

m	n	r
10	6	4
6	4	2
4	2	0

GCD = 2

مثال تطبيق 2 //

m	n	r
20	130	
130	20	10
20	10	0

GCD =10

عدد مرات تنفيذ العبارة **(Frequency Count)** :

إن عدد مرات تنفيذ العبارة يختلف حسب عينة البيانات . حيث يوجد لدينا ما يسمى بوقت التنفيذ المفرد للعبارة (execution time for single).

Total execution time = frequency count * execution time for single

إن الوقت الكلي للتنفيذ يعتمد على العوامل التالية :

1. نوع الحاسبة (Computer type).
2. لغة البرمجة (Programming language).
3. الوقت التنفيذي الخاص لكل عبارة (Total execution time).
4. نوع المترجم أو المفسر (Compiler and interpreter).

مثال 3// افترض وجود الأجزاء البرمجية الآتية مرتبة من 1 إلى 3 كالتالي :

مثال1	--------- x=x++; ---------	Fc = 1
مثال2	For(int i=1;i<= n;i++) ---------------- x = x++;	Fc = n
مثال3	For(int i=1;i<= n;i++) For(int j=1;j<= n;j++) x=x++;	Fc =n^2

لو افترضنا إن (n = 10) فان مثال رقم (2) فيه عدد مرات تكرار الخطوة التنفيذية هو(10) وعدد المرات في المثال رقم (3) هو (100).
نستنتج من ذلك إن المثال رقم (1) ينفذ أسرع من المثالين (2)و(3)ومثال (2) أسرع من مثال (3).

3-1: الوقت الكلي لتنفيذ الخوارزمية (Execution Time):

تنظيم الترتيب للخوارزمية (Order of magnitude of Algorithm) :
هو مجموع تكرارات جميع العبارات التنفيذية التي بموجبها يحدد التقدير المسبق لوقت تنفيذ الخوارزمية.
إن العبارة الغير تنفيذية تعني العبارة التي ليس لها تأثير على البرنامج مثل عبارة التعليق في أي لغة برمجية يمكن استخدامها.

مثال// لدينا مصفوفة A_{n*n} ،أحسب مجموع كل صف واخزن قيمته في مصفوفة اسمها S ، ثم احسب المجموع الكلي لعناصر المصفوفة A ، ثم اعطي عدد تكرارات المرات .

$$Sum=\sum_{\substack{i=1\\j=1}}^{n} aij$$

الحل // توجد طريقتين:

```
1- Grandtotal = 0;
For(int k =1;k<=n;k++)
{ s[k]=0;
For(int j =1;j<=n;j++)
{ s[k]= s[k]+a[k,j];
Grand total =Grand total +a[k,j];
}

}
```

نلاحظ ان عدد التكرارات يساوي 2n*n

```
2- Grandtotal = 0;
For(int k =1;k<=n;k++)
{ s[k]=0;
For(int j =1;j<=n;j++)
{ s[k]= s[k]+a[k,j]; }
Grand total =Grand total +s[k];
}
```

وهنا نلاحظ أنها تساوي n^2+n

مثال // إذا كانت لدينا خوارزمية CN وخوارزمية ثانية CN^2 علماً إن C هي ثابت ، قم بعمل مقارنة بين الخوارزميتين من حيث وقت التنفيذ علماً إن:
C الأولى =10 ، والثانية = 0.5
والـ n = { 1,5,10,15,20,25,30}

n	CN	CN^2
1	10	0.5
5	50	12.5
10	100	50
15	150	112.5
20	200	200
25	250	312.5
30	300	450

ملاحظة // إذا كانت قيمة n اقل أو تساوي 20 فان وقت الخوارزمية الثانية اقل من وقت الخوارزمية الأولى لأنه عدد مرات التكرار أو العمليات اقل، لكن بعد هذه القيمة (20) أي (25,30) فان وقت الخوارزميه الأولى يكون اقل ،هذه المرة (ينتج العكس)

عندما نقوم باحتساب وقت التنفيذ للخوارزمية معنى ذلك إننا نجد (O(g(n)) وهذا يعني إن وقت التنفيذ لن يستغرق أكثر من (C*g(n)) حيث إن (C) هو كمية ثابتة والـ n تمثل عدد العمليات المطلوبة بموجب الخوارزمية لمدخلات حجمها m .

مثلاً:

O(n) لها صيغ :

1. O(1) معنى ذلك إن وقت الاحتساب ثابت مثل (x=x++) ضمن البرنامج .
2. O(n) وهي إن وقت الاحتساب ذو صيغة خطية مثل طباعة جميع عناصر مصفوفة حجمها n أو مثلاً إيجاد عنصر في قائمة موصولة .
3. Quadratic: (وقت تربيع $O(n^2)$) مثل وقت ترتيب عناصر قائمة باستخدام عناصر الترتيب الفقاعي .
4. $O(n^3)$: هي الوقت اللازم لجعل عناصر مصفوفة (n*n*n = 0) .
5. $O(2^n)$: يعني إن الوقت اللازم لجعل جميع عناصر مصفوفة مثلاً مساوياً للصفر هو استخدام الصيغ الأسية .
6. O(log(n)) : هذه الصيغة تمثل وقت التنفيذ باستخدام الصيغ اللوغارتيمية مثال ذلك الوصول إلى عقدة نهائية في شجرة ثنائية.

ملاحظة// قيمة الـ log دائماً تكون محصورة بين (0)و(1) أي أقسام عشرية .

مثال// لديك القيم التالية لـ n ={1,2,4,8,16}
سنقوم بتطبيقها على الخوارزميات التالية ومقارنتها بجداول لغرض توضيحها :

n	Log_2n	$Nlog_2n$	N^2	N^3	2^n
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	1096	65536

$$Log_2 (x) = Log_{10} (x) *3.322$$

تمرين // ارسم الدوال السابقة ووضح عملية الفرق بين الخوارزميات.

ملاحظة1// وقت O(n) ووقت O(n Log n) يزداد كل منهما بصورة أبطأ من الدوال الأخرى .
ملاحظة2// عندما يكون حجم البيانات كبير تصبح الخوارزمية لها تنفيذ وقت كبير فمثلاً الخوارزمية التي عدد عملياتها $O(nLog_2n)$ تكون غير واقعية أو غير عملية .
ملاحظة3// الخوارزمية التي وقت تنفيذها بالصيغة الأسية بالإمكان اعتمادها فقط إذا كانت قيمة الـ (n) صغيرة، حيث إذا كانت قيمة الـ n صغيرة فإن عدد العمليات قليل وبالتالي فإن وقت التنفيذ أسرع وبالتالي التغير سيكون واضح.

مثال// لديك الدالة التالية

$$F=\frac{a+b^2+c+(a+b)}{(a+b)+4.0}$$

علماً إن الثوابت (a=2.0,b=3.0,c=5.0) ،
المطلوب// تحديد تعقيدات الوقت وتعقيدات الخزن علماً إن (a,b,c) هي قيم حقيقية.

ملاحظة// عند التعويض بالمعادلة $\frac{21}{9} = 2.33$ ،الخزن يعني هنا ما هي المتغيرات الموجودة في السؤال ؟

تعقيدات الخزن : توجد لدينا خمسة خلايا خزنيه هي a,b,c وعنوان العودة F ومتغير لاسم الدالة ، معنى ذلك إن $S_{a,b,c}$ (a,b,c)= ϕ أي لايوجد جزء متغير هنا أي انه جزء ثابت .

ففي حالة إن يكون المطلوب إعادة هذه الدالة n من المرات ما هي تعقيدات الخزن هنا ،علماً إن n=3

والـ (a,b,c) قيمها كالتالي:

n	a	b	c
3	2.1	9.3	2.0
2	3.2	4.2	3.0
1	4.1	3.3	5.0

للحل نقوم بالتالي :

نقوم بعمل جدول كالسابق بمعنى ذلك انه توجد قيم متغيرة للمتغيرات فالنتيجة لا تساوي صفر وتعتمد على قيم المتغيرات .

تعقيدات الوقت :

في الحالة الأولى وقت الدالة يساوي واحد (Count=1) لأنها تنفذ مرة واحدة ،وفي الحالة الثانية الوقت يعتمد على عدد المرات (Count=n) .

مثال// لإيجاد مجموع عناصر مصفوفة أحادية البعد كما في المعادلة التالية:

$$Sum = \sum_{i=1}^{n} ai$$

```
Float Sum(float  a[ ],int n)
{ float S=0.0 ;     ------(1= عدد الخطوات)
For(int i=1;i<=n;i++)  -----(n+1=عدد الخطوات)
S+=a[2];------(n)
Return S------(1)
}
```

يتضمن مثال المسألة بالمتغير (n):

تعقيدات الخزن : تتطلب الدالة (Sum) ستة خلايا خزنيه لخزن قيم (I,S,n) وعنوان المصفوفة a[] والمتغير الذي يحمل اسم الدالة إضافة إلى عنوان العودة).

وهو خزن ثابت لا يعتمد على خصائص المثال (أي لا يعتمد على تغير قيمة n).

$S_{sum}(n)=0$

تعقيدات الوقت :

$T_{sum}(n)=2n+3$

نلاحظ إن العلاقة التي تربط الوقت بعدد العناصر هي علاقة خطية وهي أفضل من التربيعية .

مثال// سوف نأخذ نفس المثال السابق ولكن هذه المرة بطريقة الاستدعاء الذاتي (Recursion)

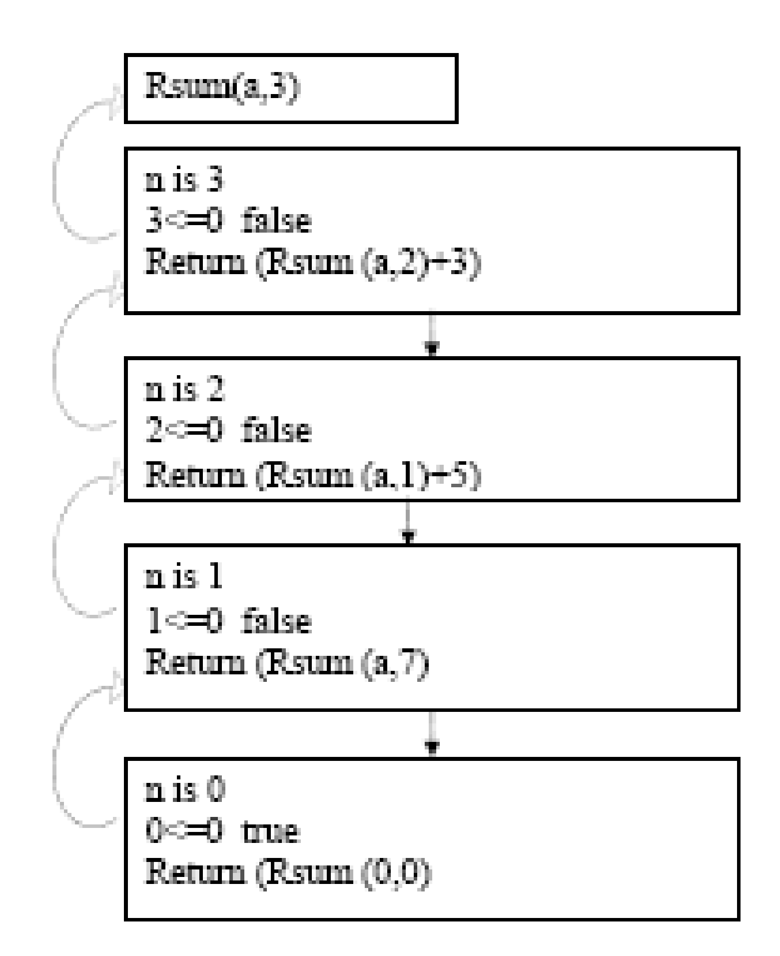
$$Sum = \sum_{i=1}^{n} ai$$

$$Rsum(a,n) = \begin{cases} 0,0 & if\ (n <= 0) \\ Rsum(a,n-1) + a(n) & if\ (n > 0) \end{cases}$$

هذا بالنسبة للصيغة الرياضية أما بالنسبة للصيغة البرمجية فهي :

```
Float Rsum(float a[],int n)
{if  (n<=0)  return (0.0);
  Else return (Rsum(a,n-1)+a[n]);
}
```

ولنفترض إن المصفوفة (a[1..3]=3,5,7) أي إن (n=3)



تعقيدات الفراغ :

يتضمن فراغ مكدس التداخل المعاملات الشكلية والمتغيرات المحلية وعنوان العودة.
كل تنشيط (استدعاء) يتطلب أربع خانات خزنيه (خلية بقيمة (n) وخلية للمؤشر إلى المصفوفة (a) وخلية للمتغير الذي يحمل اسم الدالة بالإضافة إلى خلية لعنوان العودة) .
وبما إن عمق التداخل(عدد الاستدعاءات) يحسب كالتالي:

عمق التداخل (الاستدعاء) = | الحجم الأولي في أول تنشيط - الحجم النهائي في أخر تنشيط |+ 1

عمق التداخل (الاستدعاء) = | n − 0 | +1= 4

حيث إن الحجم الأولي في أول تنشيط نقصد به عدد العناصر n ثم يأخذ بالتناقص ،أما الحجم النهائي في أخر تنشيط فممكن إن يكون (0) أو أية قيمة أخرى.

$T_{rsum}(n)=4(n+1)$

من المعادلة أعلاه نلاحظ إن الخزن دالة خطية يعتمد على قيمة الـ(n) حيث إذا ازدادت قيمة (n) ازداد الخزن أما إذا قلت قيمة الـ (n) قل الخزن

تعقيدات الوقت : سوف نقوم باستخدام علاقات التداخل (Recurrence relations) لحسابها كالتالي:

$$t_{rsum}(n) = \begin{cases} 2 & if \quad n <= 0 \\ 2 + t_{rsum}(n-1) & if \quad n > 0 \end{cases}$$

إن القيمة (2) تمثل الزمن المطلوب لاستدعاء كل تنفيذ ، أما ($t_{rsum}(n-1)$) فهي تمثل وقت الدالة بدون أخر استدعاء.

إن خطوة (Else) لا تعتبر خطوة معالجة لذلك فهي تأخذ القيمة (0) ولكن الخطوة التي تحويها تأخذ القيمة (1).

ولحل هذه العلاقة التداخلية نستخدم طريقة من طرق الحل وهي طريقة التعويض التكراري (Iterative Substitution) كالتالي :

$$\begin{aligned} t_{rsum}(n) &= 2 + t_{rsum}(n-1) \\ &= 2 + 2 + t_{rsum}(n-2) \\ &= 2(2) + t_{rsum}(n-2) \\ &= 2(2) + 2 + t_{rsum}(n-3) \\ &= 3(2) + t_{rsum}(n-3) \\ &= m(2) + t_{rsum}(n-m) \end{aligned}$$

وعندما (m=n) فإن :

$$= 2n + t_{rsum}(n - n) \quad , \quad n>0$$
$$= 2n + t_{rsum}(0)$$
$$= 2n + 2$$

ليس بالضرورة إن يكون لدينا (n-n) فقد يكون هناك (n-1) أو أي قيمة أخرى حيث إن المهم إن يكون لدينا (n-m).

مثال// لإيجاد مجموع مصفوفتين ثنائيتين كما في المعادلة التالية :

$$C_{m*n} = A_{m*n} + B_{m*n}$$

```
Void  Add(type  a[][size], type  b[][size], type  c[][size],int m,int n)
{ for(int i=1;i<=m;i++)    ........m+1
  For (int j=1;j<=n;j++)    .......Sm
     C[i][j]=a[i][j]+b[i][j];
}
```

نلاحظ هنا إن مثال المسألة يتصف بالمتغيرات (m,n) حيث :

تعقيدات الفراغ : تتطلب الدالة (Add) ثمان خانيا خزنيه لخزن قيم المتغيرات (m,n,a,b,c,i,j) بالإضافة إلى عنوان العودة، ونلاحظ إن الخزن لا يتوقف على خصائص المثال أي إن :

$$S_{add}(m,n) = 0$$

أما بالنسبة إلى تعقيدات الوقت : فهي كالتالي

$m + 1$

Sm

$(n + 1 + n)m$

$(2n + 1)m$

$2nm + m + m + 1$

$T_{add}(n,m) = 2nm + 2m$

إن هذه العلاقة تكون مقبولة في حالة (m<=n) ، أما عندما تكون (m>n) فانه يفضل إبدال تعليمتي الـ (For) لان ذلك يحفظ تعقيدات الوقت لتصبح :

$$T_{add}(n,m) = 2nm + 2n + 1$$

مثال// اوجد تعقيدات الخزن والوقت لسلسلة أعداد فيبوناتشي (Fibonatia) التي هي متتابعة من الإعداد تبدأ كما يلي:
أول حدين فيها هما (0,1) وهما ثابتان في المتتابعة حيث إن عملية الحساب تتم على الحدود الباقية من خلال جمع الحدين السابقين كالتالي :
0,1,1,2,3,5,8,13,……

حيث إن كل حد جديد يتم الحصول عليه من خلال جمع الحدين السابقين له فإذا كان (F0) يمثل الحد الأول في المتتابعة فانه وبصورة عامة :

$$F0 = 0$$
$$F1 = 1$$
$$Fn = (Fn-1) + (Fn-2), n >= 2$$

إن طريقة عمل أو تنفيذ البرنامج تتم بإدخال عدد صحيح موجب وليكن (n) وتطبع قيمة (Fn) له حيث إذا كانت (n=3) فان (Fn=2) أو (n=4) فان (Fn=3).
إن جزء البرنامج الخاص بالمتتابعة هو:

```
Void Fibonacci (int n)
{ // Compute the nth Fibonacci number.
  If (n<=1)            …………+1
    Cout<<n<<endl;      ……+1
  Else
    {  int Fnm1=0,Fnm2=1,Fn;      ….+2
       For(int i=2;i<=n;i++)      …..+n
        { Fn=Fnm1+Fnm2;           …..n-1
          Fnm1=Fnm2;              …..n-1
          Fnm2=Fn;               ……n-1
        }
     Cout <<Fn<<endl;            …….+1
}
```

إن خصائص هذا المثال تتصف بالمتغير (n).

تعقيدات الخزن : يتطلب البرنامج ستة خلايا خزنيه بخزن قيم المعاملات (Fnm1,Fnm2,Fn,I,n) وعنوان العودة وهو خزن ثابت لا يعتمد على خصائص المثال أي إن :

$$S_{fibonacci}(n) = 0$$

تعقيدات الوقت :
يجب هنا اعتبار حالتين لتحليل تعقيدات الخزن للبرنامج وذلك لوجود شرط كالتالي :
الحالة الأولى : عندما (n=0,n=1) فان تعقيدات الوقت (عدد الخطوات) هي (2).
الحالة الثانية : عندما (n>1) فان عدد الخطوات هو (4n+1) وكما يلي :

$$T_{fibonacci}(n) = \begin{cases} 2 & if \quad n \in (0,1) \\ 4n+1 & if \quad n > 1 \end{cases}$$

ولنقم بعملية حساب عدد الخطوات في الأجزاء البرمجية التالية:

```
int  i = 1;
while  (i <=  n)
{ x + + ;
i + + ;
}
```

نلاحظ إن الأداة (while) تأخذ (n+1) من الخطوات مع مراعاة الانتباه إلى بداية العداد المستخدم (i) والزيادة المعطاة له .

إما في جزء (Do while) هذا فنلاحظ إن (while) والعبارات الخاصة بها تأخذ (n) من العمليات.

```
int   i = 1;
Do  {
x + +;
i + +;
} while  (i <=  n);
```

مثال // اوجد تعقيدات الخزن والوقت لمسألة حساب ما يسمى بالمتوسطات السابقة (Prefix Average) لمتتابعة من الإعداد .

يمكن توضيح المسألة كالآتي: إذا كان لدينا مصفوفة معينة ولتكن (X) مخصصة لخزن (n) من الإعداد الصحيحة،فان المطلوب حساب مصفوفة معينة هي (A) حيث إن العنصر (Ai) يمثل متوسط قيم العناصر من ([X[0]……X[i) لقيم (i=0,1,…,n-1) أي انه :

$$A[i] = \frac{\sum_{j=0}^{n} x[i]}{i+1}$$

Algorithm Prefix Averages(x):

Input: An n-element array x of number.
Output: An n-element array A of number.
That A[i] is the Average of elements X[0],……,X[i].

$For\ \ i \longleftarrow 0\ \ to\ \ n-1\ \ do\ \ .... n+1$

$a \longleftarrow 0\ \ .... n$

$For\ \ j \longleftarrow 0\ \ to\ \ i\ \ ....\sum_{i=1}^{n}(i+2)$

$a \longleftarrow a + x[j]\ \ ....\sum_{i=1}^{n}(i+1)$

$and\ \ \ for\ \ j$

$A[i] \longleftarrow a/(i+1)......... ..n$

$and\ \ \ for\ \ i$

$return\ \ \ arrayA\ \ \ .... n$

تعقيدات الخزن: تتطلب المسألة ستة خانيا خزنيه لخزن قيم المعاملات (a[],x[],n,I,j) وعنوان العودة وهو خزن ثابت لا يعتمد على خصائص المثال أي إن :

$$S_{prefixaverages}(n) = 0$$

تعقيدات الوقت :

$$\sum_{i=1}^{n}(i+2) = \sum_{i=1}^{n} i + \sum_{i=1}^{n} 2$$

$$= \frac{n(n+1)}{2} + 2 * \sum_{i=1}^{n} 1 =$$

$$= \frac{n(n+1)}{2} + 2 * n = \frac{n(n+5)}{2}$$

أما بالنسبة إلى

$$\sum_{i=1}^{n}(i+1) = \frac{n(n+3)}{2}$$

وتحليل العملية :

$$1+2+3+..........+n-2\ \ +n-1\ \ +n$$

إن تعقيدات الوقت هي :

$$T_{prefixaverages}(n) = n^2 + 7n + 1$$

ملاحظة // إن (begin,end,else,…etc) تعتبر موجهات للمترجم ولا تأخذ أي خزن أو وقت للتنفيذ (عبارات توجيهية).
نلاحظ إن تعقيدات الوقت لهذه الخوارزمية أصبحت تربيعية فهل يمكن تحويلها إلى خطية ؟
نحاول ذلك كما في الآتي :

4-1: الحالات الأفضل والأسوأ والمتوسطة للتحليل
(Best & Worst & Average Cassese Analysis)

يجب علينا إن نلاحظ فيما إذا كانت المسألة تأخذ أكثر من حالة ،وسوف نقوم بالتركيز على الحالة الأسوأ للمسائل لأنها تحوي تعقيدات كثيرة، كما يمكننا التخلص من الصعوبات في الحالات التي تكون فيها المعاملات المختارة(خصائص المثال) غير مناسبة أو كافية وحدها لتحديد عدد الخطوات وذلك من خلال تحديد ثلاث أنواع من الخطوات:

أولا: عد خطوات الحالة الأفضل وهو ادني عدد من الخطوات يمكن تنفيذها لمعاملات معينة .
ثانياً : عد خطوات الحالة الأسوأ وهو أقصى عدد من الخطوات يمكن تنفيذها لمعاملات معينة.
ثالثاً: عد خطوات الحالة المتوسطة وهو العدد المتوسط من الخطوات التي يمكن تنفيذها على أمثلة مسألة بمعاملات معينة.

مثال// اوجد العنصر الأكبر في مصفوفة أحادية البعد؟

```
Algorithm Arraymax (A,n):
Input: An array A storing  n  integer.
Output: the maximum element in A.

a[0] ←—— CurrentMax
1  to  n-1  do ←—— For i
  If  CurrentMax <A[i]  then
A[i] ←——       CurrentMax
  Endif
Endfor
Return CurrentMax
```

- الحالة الأفضل: يكون البحث في أفضل حالاته عندما يكون أول عنصر في المصفوفة هو الأكبر حيث لا يدخل في تنفيذ إيعاز (if) .
- الحالة الأسوأ: يكون البحث في أسوأ حالاته عندما يكون أخر عنصر في المصفوفة هو الأكبر حيث سيتم تبديل القيمة حتى الوصول إلى النهاية .
- الحالة المتوسطة : حيث نلاحظ تعقيدات الحالة هي :

(عدد الخطوات لحد الوصول إلى العنصر (n.......3,2,1) \ عدد الخطوات الكلية (n))

كما نلاحظ فان مستوى التعقيدات يكون حسب الحالة ففي الحالة الأفضل تكون اقل تعقيدات وفي الحالة الأسوأ تكون اعلى تعقيدات .

ملاحظة// إن طريقة حساب عدد الخطوات (Step Count) هي طريقة حساب تقريبية وليس دقيقة وهي صعبة بنفس الوقت.

$$T_{ArrayMax}^{B}(n) = 2n + 1$$

$$T_{ArrayMax}^{W}(n) = 3n$$

$$T_{ArrayMax}^{A}(n) = \frac{\sum_{i=1}^{n}(2n+i)}{n} = \frac{2n + \sum_{i=1}^{n} i}{n} = \frac{2n + \frac{n(n+1)}{2}}{n}$$

مع ملاحظة انه في الحالة المتوسطة فانه يجوز للعداد (i) إن يبدأ من الصفر أو الواحد ، أما إذا كانت عملية الحساب تبدأ بالعكس أي من النهاية إلى البداية فتكون :

$$T_{ArrayMax}^{A}(n) = \frac{\sum_{i=0}^{n}(3n - i + 1)}{n}$$

5-1: الصيغ التقاربية (Asymptotic notation):

يوجد ثلاث صيغ تقريبية هي :

1. صيغة الحد الأعلى (Big-Oh).
2. صيغة الحد الأدنى (Omega).
3. صيغة الحد الأعلى – الحد الأدنى (Theta).

برهنت عملية تحديد عد الخطوات (Steps Count) على أنها مهمة غاية في الصعوبة لذلك احتجنا إلى صيغ تقريبية لتحديد هذه الخطوات :

1. صيغة الحد الأعلى (Big-O) :

ويقصد بها إن تعقيدات الخزن أو الوقت ممكن إن تساوي الحد الأعلى أو تكون اقل منه ولا يمكن إن تكون اعلي منه وعملية الصياغة تتم كالآتي :

$$f(n) = C \quad g(n)$$

هذه المعادلة تطبق إذا وفقط إذا وجد ثابتان موجبان هما (C, n_0) بشرط إن ($C>0 \quad \& \quad n_0>=1$) حسب التنفيذ التالي :

$$f(n) \le Cg(n)$$

$n \ge n_0$ لجميع قيم

نظرية :إذا كانت

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots\dots\dots\dots + a_1 n_1 + a_0$$

متعددة حدود درجتها (m) فان :

$$f(n) = O(n^m)$$

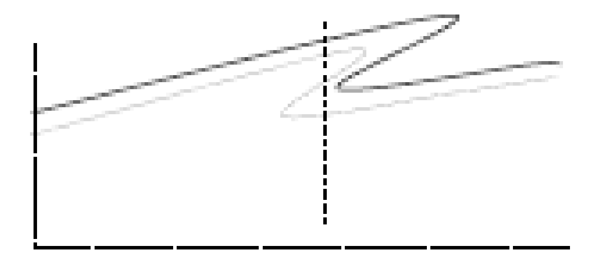
وهذه بعض الأمثلة لتطبيق النظرية :

مثال1// إذا كانت $3n + 2$ تمثل F(n) (تعقيدات خزن ووقت) لأي برنامج معين فان :
الحل //

$$3n + 2 = O(n)$$

لان $3n + 2 \le 4n$
لجميع قيم $n \ge 2$

حيث إن الكمية (4 n) تمثل الثوابت أي إن n هي الثابت g(n) بينما 4 تمثل الثابت c ، حيث إننا أخذنا اكبر معامل للـ n وهو(3) وأضفنا له واحد ليصبح (4) كما في علاه بإن $O(n)$ تمثل الحد الأعلى لتعقيدات البرنامج .
وهذا يعني أنها عملية إهمال التفاصيل (النقاط الحادة) في منطقة معينة من دالة مع البقاء على شكل الدالة بدون تأثير كما في الشكل رقم (2) التالي :



شكل(2):صيغة O _Notation

- عملية التصاعد منتظمة
- على الأقل يوجد عملية واحدة

ملاحظة // في الرسوم نستخدم الشاشة تبدأ من الإحداثيات (0,0) إلى MaxX,MaxY لأننا نأخذ الجزء الموجب فقط .

Example1: Use Big_O Notation to analyze the time efficiency of following c++ code of the integer N .

```
For(int i=1;i<= N/2;i++)
{-----
  For(int j=1;j< = N*N;j++)
   {-----
    -----
   }
}
```

نلاحظ إن دوارة For( ) الخارجية تنفذ (N/2) بينما For( ) الداخلية تنفذ (N*N) كالتالي :

$N/2 * N*N$

$N/2 * N^2$

$½ N * N^2 = \text{big-O} = O(N^3)$

```
EX : N=5;
      for(int k=1 ;k<=2;k++)
        for(int j=1;j<=25;j++)
          {}
```

Than O(125) =

k	J
1	1
	.
	.
2	25
1	1
	.
	.
2	25

Example2: Use Big-O Notation to analyze the time efficiency of following c++ code of the integer n .

```
For(int i=1;i<= n/2;i++)
{-----}
For(int j=1;j< = n*n;j++)
{-----}
```

$n/2 + n*n = 1/2n + n^2 = n+n^2$

$n(1+n) = \text{Big-O} = O(n^2)$

ملاحظة// الكمية (1/2) و(1) يهملان باعتبارهما كمية ثابتة يرمز لها (C) ،وصيغة (Big-O) لا تحتاج إلى ثوابت ، فلو فرضنا مثلاً:

```
N=5 ;
For(int i=1;i<= 2;i++)
For(int j=1;j< = n*n;j++)
```

هنا نحتاج (27) تكرار فقط للدورات وهذا يعني إن حالة التنفيذ هنا هي أسوأ من الحالة العامة .

مثال// افترض انه لديك المقطع التالي:

```
k=n;
Do
{---------
 K=k/2;
} While (k >1);
```

هنا عندما تنقص قيمة الـ(n) فهذا يعني انه الدالة أما n Logn أو Log n بينما لا يمكن إن تكون $O(n^2)$. $O(n)$ هي $O(n^3)$.$O(n^2)$.

When n=8;
Then k=8;

k	العملية
8	8>1
4	4>1
2	2>1
1	1=1

إن عدد مرات تكرار هذا المقطع يتناقص إلى النصف في كل مرة لذا فانه يتمثل بالـ Log_2n أي إن (Big-O) له هي $O(Log_2n)$ وتتجاهل $O(n^3)$,$O(n^2)$,$O(n)$, $O(2^n)$ وذلك لانها تزيد من قيمته وتجاهلنا nLogn لانها تضرب في n .

كما يمكن صياغة المثال بالصورة التالية :
لديك المتعددة التالية :

$$3n + 2 \leq 4n$$

المطلوب// إيجاد الصيغة التقريبية لها ثم إيجاد قيمة (n).

الحل// أي متعددة يكون شكلها كالتالي :

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots\dots\dots\dots + a_1 n_1 + a_0$$

بما إن المتعددة فيها اقل أو يساوي هذا يعني إن الصيغة هي الأولى (Big-O) وتعتمد على n ،وبما إن اعلى قيمة للـ(n) هو الواحد وبالتالي يكون الحل بالصيغة الأولى :

$$3n + 2 = O(n)$$

لان $3n + 2 \leq 4n$
لجميع قيم $n \geq 2$

n	الطرف الأيسر	الطرف الأيمن	$3n + 2 \leq 4n$	التحقق
1	5	4	$4 \leq 5$	False
2	8	8	$8 \leq 8$	True
3	11	12	$12 \leq 11$	True
4	14	16	$16 \leq 14$	True

∴ الصيغة تتحقق عندما قيمة الـ (n) اكبر من أو تساوي (2)

مثال2// إذا كانت $10n^2 + 4n + 2$ تمثل F(n) (تعقيدات خزن ووقت) لبرنامج معين اوجد التعقيدات بدلالة الصيغ التقاربية :

$10n^2 + 4n + 2 = O(n^2)$

لان $10n^2 + 4n + 2 \le 11n^2$

لجميع قيم $n \ge 5$

كما يمكن صياغة السؤال بالصورة التالية:
لديك المتعددة التالية :

$$10n^2 + 4n + 2 \le 11n^2$$

ما هي الصيغة المستخدمة وقيمة الـ (n).

الحل// بما إن المتعددة اقل أو تساوي فان الصيغة هي (Big-O) واعلي أس للـ (n) يمثل أس الـ(n) في الصيغة .

$10n^2 + 4n + 2 = O(n^2)$

$10n^2 + 4n + 2 \le 11n^2$

لجميع قيم $n \ge 5$

n	الطرف الأيسر	الطرف الأيمن	$10n^2 + 4n + 2 \le 11n^2$	التحقق
1	16	11	$11 \le 16$	False
2	50	44	$44 \le 50$	False
3	104	99	$99 \le 104$	False
4	178	176	$176 \le 178$	False
5	272	275	$275 \le 272$	True
6	286	396	$396 \le 286$	True
7	786	847	$847 \le 786$	True

مثال3// إذا كانت $100 = O(1)$ اوجد التعقيدات بدلالة الصيغ التقاربية ؟

الحل// إن قيمة 100 تمثل F(n) ،والثابت g(n) يمكن تمثيله بالقيمة (1) ، هذا يعني:

$100 \le 100 * 1$

لجميع قيم $n \ge 1$

مثال4// $6 * 2^n + n^2 = O(2^n)$ اوجد تعقيدات الخزن والوقت بدلالة الصيغ التقاربية ؟

الحل// نلاحظ إن هذا البرنامج يملك تعقيدات آسية وهي اعلي تعقيدات لذلك فان :

$$6 * 2^n + n^2 = O(2^n)$$

$6 * 2^n + n^2 \le 7 * 2^n$ لان

$n \ge 4$ لجميع قيم

ملاحظة// إذا وضعنا قيمة (n)g بحيث تكون اعلى من ما موجود في الأمثلة السابقة فهو لا يؤثر رياضياً أي تبقى العلاقات صحيحة .

مثال5// تحقق من صحة المعادلة $10n^2 + 4n + 2 \ne O(n)$ ؟

الحل//

$$10n^2 + 4n + 2 \ne O(n)$$

$10n^2 + 4n + 2 \le 10^6 n$ لان

لجميع قيم $n \ge 10^6$ وهذا غير ممكن .

مثال6// تحقق من صحة المعادلة $3n + 2 \ne O(1)$ ؟

الحل//

$$3n + 2 \ne O(1)$$

$3n + 2 \le 10^6 * 1$ لان

لجميع قيم $n \ge 10^6$ وهذا غير ممكن

إذ يجب إن تكون $n \le 10^6$.

2- صيغة الحد الأدنى (Omega Ω):

ويقصد بها إن تعقيدات الخزن أو الوقت ممكن إن تكون اكبر أو تساوي الحد الأدنى ولا يمكن إن تكون اقل منه وعملية الصياغة تتم كالآتي :

$$F(n) = \Omega(c \ g(n))$$

إذا وفقط إذا كان لدينا ثابتان موجبان هما (C, n_0) بحيث ($C>0$) و ($n_0>=1$) فان :

$$F(n) \ge cg(n)$$

لجميع قيم $n \ge n_0$

نظرية :إذا كانت

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n_1 + a_0$$

متعددة حدود درجتها (m) فان :

$$F(n) = \Omega(n^m)$$



شكل(3):صيغة Ω _Notation (اوميكـا)

- عملية التصاعد غير منتظمة (مبعثرة)
- يوجد زاوية (0) ولها قيمة
- ربما المدى يبدأ من 0 مثلاً [0,2] ومثاله استخدام أي زاوية فانه سيحول أو يغير شكل الدالة تماماً.

مثال1//إذا كان $3n + 2$ تمثل تعقيدات خزن ووقت لبرنامج معين اوجد هذه التعقيدات بدلالة الصيغ التقاربية ؟
الحل //

$3n + 2 = \Omega\, 3n$
لان $3n + 2 \geq 3n$
لجميع قيم $n \geq 1$

لاحظ هنا معامل n وهو (3) يبقى كما هو أي إن الثابت c =3 ، كما يمكن تمثيل الصيغة $3n + 2 = \Omega(1)$ فإنها لا تؤثر وتبقى الصيغة صحيحة.

كما يمكن صياغة السؤال بالصورة التالية:
لديك المتحددة التالية :

$3n + 2 \geq 3n$

المطلوب// إيجاد الصيغة التقريبية ثم إيجاد قيمة الـ (n).

$3n + 2 = \Omega\, 3n$
لان $3n + 2 \geq 3n$
لجميع قيم $n \geq 1$

n	الطرف الأيسر	الطرف الأيمن	$3n + 2 \geq 3n$	التحقق
1	5	3	$5 \geq 3$	True
2	8	6	$8 \geq 6$	True
3	11	9	$11 \geq 9$	True

مثال2// $3n + 2 = \Omega(n)$ فما هي المتعددة ؟
الحل//بما إن الصيغة هي Ω هذا يعني إن العلاقة هي $\geq$ فتكون المتعددة :

$$3n + 2 \geq 3n$$

مثال3// لديك الصيغة التقريبية التالية :

$$10n^2 + 4n + 2 = \Omega(n^2)$$

المطلوب// تكوين متعددة لهذه الصيغة علماً إن الـ (C=11)
الحل// بما إن الصيغة هي للحد الأدنى هذا يعني إن الثابت يجب إن يكون اقل أو مساوي للحد الأدنى أي انه:

$$10n^2 + 4n + 2 \geq 10n^2$$

لجميع قيم $n \geq 1$

مثال4// اثبت صحة هذه المعادلة $6 * 2^n + n^2 = \Omega(2^n)$ ؟
الحل//

$$6 * 2^n + n^2 = \Omega(2^n)$$

لان $6 * 2^n + n^2 \geq 6 * 2^n$

لجميع قيم $n \geq 1$

3- صيغة الحد الأعلى_الأدنى (Theta Θ):

تستخدم الصيغة التالية حسب المتعددة التي كتبناها سابقاً :

$$F(n) = \Theta(g(n))$$

إذا وفقط إذا وجدت الثوابت الموجبة التالية (n_0, C_1, C_2) بحيث يكون :

$$C_1 g(n) \leq F(n) \leq C_2 g(n)$$

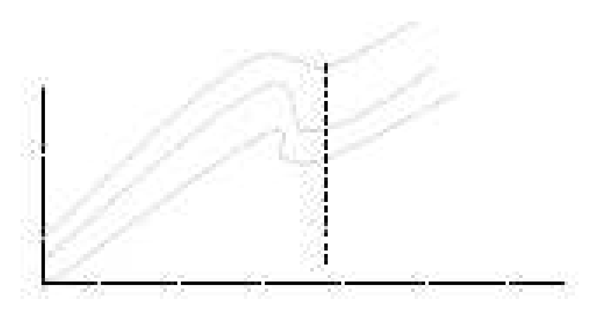
لجميع القيم $n \geq n_0$

نظرية :إذا كانت

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots\ldots\ldots\ldots + a_1 n_1 + a_0$$

متعددة حدود درجتها (m) فان :

$$F(n) = \Theta(n^m)$$



شكل(4):Θ _Notation (ثيتــا)

- كل زاوية تغير الشكل تماماً
- نفس الطول وكل مرة تختلف الزاوية

مثال1// اثبت صحة المعادلة $3n + 2 = \Theta(n^2)$ المعطاة بدلالة صيغة تقاربية ؟

الحل// المعادلة صحيحة

$$3n + 2 = \Theta(n^2)$$

لان $3n \le 3n + 2 \le 4n$

لجميع قيم $n \ge 2$

لاحظ هنا إن تحديد قيمة الثابت (2) يكون تجريبي وليس عشوائي ،أي انه (2) فما فوق يحقق الصيغة بينما القيمة (1) لا تحقق الصيغة.

مثال2// طبق الصيغة التالية $3n \le 3n + 2 \le 4n$

الحل// بما انه توجد قيم عليا وقيم سفلى ،هذا يعني انه يجب استخدام صيغة الثيتا Θ

$$F(n) = \Theta(g(n))$$

وبما إن الثوابت (C1,C2) اكبر من الصفر هذا يعني تحقق الشرط الأول ،لذلك نقوم بتطبيق الجدول:

n	الوسط	الطرف الأيسر	الطرف الأيمن	التحقق
1	4	5	3	False
2	8	8	6	True
3	12	11	9	True
4	16	14	21	True

نستنتج إن الحل الصحيح يبدأ من $n \ge 2$

مثال3// بين إن الصيغة التالية صحيحة :

$$10\ n^2 + 4n + 2 = \Theta(n^2)$$

الحل// إن اعلي قيمة للـ n تضرب بالثوابت .

$$10\ n^2 \leq 10\ n^2 + 4n + 2 \leq 11\ n^2$$

لجميع قيم $n \geq 5$

نلاحظ انه يجب إن نضع اعلي قيمة للـ(n) في الجهتين والقيمة الأقل للـ(C) توضع في الجهة اليسرى والأكبر في الجهة اليمنى .

n	الطرف الأيسر	الأوسط	الطرف الأيمن	التحقق
1	10	16	11	False
2	40	50	44	False
3	90	104	99	False
4	160	178	176	False
5	250	272	275	True

ملاحظة// إن صيغة الحد الأعلى_ الأدنى هي الصيغة الأكثر دقة وتتحقق عندما يكون g(n) هو الحد الأعلى والأدنى للدالة F(n) .

تمرين1// برهن إن العلاقات التالية صحيحة :

$$5\,n^2 - 6\,n = \Theta(n^2) \qquad (C1=5, C2=11)$$

$$38\ n^3 + 4\,n^2 = \Omega(n^3) \qquad (C=7)$$

تمرين 2// برهن إن العلاقات التالية غير صحيحة :

$$10\ n^2 + 9 = O(n)$$
$$n^2 + Log_2 n = \Theta(n)$$

وهذه بعض حلول الأمثلة السابقة بطريقة الصيغ التقاربية :

$$S_{abc}(a, b, c) = \Theta(1)$$
$$T_{abc}(a, b, c) = \Theta(1)$$

بما إن صيغة الحد الأعلى تساوي صيغة الحد الأدنى فبالإمكان حينئذ استخدام صيغة الثيتا وبالعكس من ذلك فانه عند استخدام صيغة (Θ) فانه يمكن استخدام صيغة (O) أو (Ω)

$$S_{sum}(n) = \Theta(1)$$
$$T_{sum}(n) = \Theta(1)$$

$S_{add}(n, m) = \Theta(1)$

$T_{add}(n, m) = \Theta(m, n)$

$T_{add}(n, m) = \Theta(n^2)$ m=n وفي حالة

6-1: الصيغ الشائعة لأوقات التنفيذ (The Times Of Executive Notation):

يمكن توضيح الصيغ التي نستطيع من خلالها تمثيل أوقات التنفيذ بالمعادلة التالية :

$$O(1) < O(Logn) < O(n) < O(nLogn) < O(n^2) < O(n^3) < O(2^n)$$

- إن $O(1)$ تعتبر أفضل من باقي الصيغ الأخرى وهكذا بالنسبة لبقية الصيغ أي إن $O(Logn)$ هي أفضل من باقي الصيغ التي بعدها .
- إن الخوارزميات التي لها تعقيدات خزن أو وقت اكبر من $O(nLogn)$ تعد خوارزميات غير عملية وكذلك فان الخوارزميات التي تمتلك تعقيدات أسية أي $O(2^n)$ تكون مخيفة ولا تكون عملية إلا عندما تكون قيمة n صغيرة جداً أي اقل من 40 .
- ولتوضيح ذلك نفترض وجود حاسب ينفذ 10^9 تعليمة بالثانية الواحدة وعندما تكون $F(n) = O(2^n)$

When n =10 then f(n)=1 ms
When n =26 then f(n)=1 ms
When n =36 then f(n)=1 sec
When n =40 then f(n)=18.3 min
When n =50 then f(n)=13 day
$4*10^{13}$ *year* When n =100 then f(n)=
$32*10^{283}$ *year* When n =1000 then f(n)=

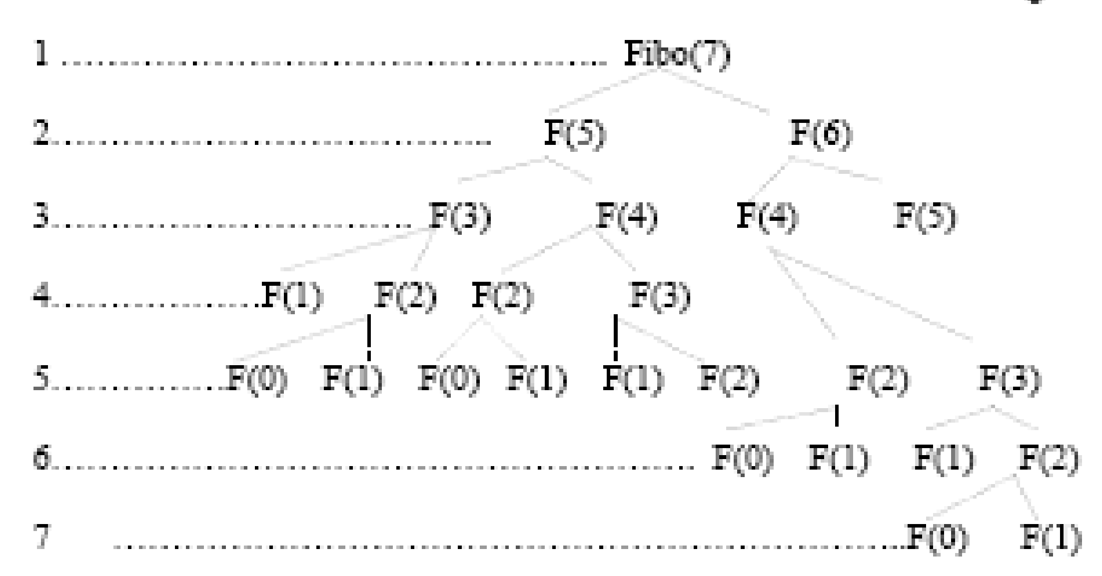
تمرين محلول// اوجد حل مسألة فيبوناتشي باستخدام أسلوب التداخل (الإستدعاء الذاتي)

$$Fibo(n) = \begin{cases} n & n < 2 \\ Fibo(n-1) + Fibo(n-2) \end{cases}$$

```
Int Fibo1 (int n)
{if (n < 2) return n;
 Else
     Return (Fibo1(n-1)+Fibo1(n-2));
}
```

تعقيدات الوقت :

إن عملية حساب التعقيدات لمسألة استدعاء ذاتي دائما نحتاج فيها إلى رسم مخطط لتوضيح تكرارات الاستدعاء ،لذلك سوف نقوم برسم شجرة تنشيطات لمسألة إعداد فيبوناتشي كما في التالي:



1 – 2^n تمثل اقصى عدد للعقد في الشجرة وبما ان كل عقدة تمثل استدعاء فان هذه الكمية تمثل عدد الاستدعاءات مع ملاحظة انه لا نحتاج الى حساب التعقيدات الخاصة بكل استدعاء .

7-1: الاستدعاء الذاتي لشجرة التنشيطات أو الاستدعاءات(**Recursion Tree**):

ملاحظة// دائماً في الشجرة الثنائية أقصى عدد من العقد هو ($2^k - 1$) حيث إن الـ k تمثل عمق الشجرة (المستوى الأقصى) لذلك فان تعقيدات فيبوناتشي تكون:

$$T_{fibo\ 1}(n) = O(2^n)$$

ولتوضيح ذلك :

عدد فيبوناتشي	عدد مرات التنشيط
7	1
6	1
5	2
4	3
3	5
2	8
1	13

نلاحظ انه لو طلبنا مثلاً Fibo1(30) فان التنشيطات سيكون عددها ما يقارب 500.000 تنشيط

تمرين // اوجد حل مسألة فيبوناتشي باستخدام أسلوب التداخل بحيث تصبح تعقيدات الوقت خطية بدلاً من أسية .

مثال// احسب تعقيدات أوقات البحوث الناجحة والفاشلة لخوارزمية البحث التعاقبي (Sequential Search) التي تتمثل في البحث عن عنصر معين في مصفوفة أحادية البعد بحيث إن هذه الدالة ممكن إن ترجع القيمة صفر إذا كان العنصر غير موجود أو ترجع قيمة موقع العنصر إذا كان موجوداً .

```
int SeqSearch ( Type a[] , Type x , int n)
{ int i=n;
  a[0]=x;
   while (a[i] != x)
    i--;
   return I;
}
```

الحل //

لحساب تعقيدات البحوث الناجحة أي إن العنصر الذي نبحث عنه موجود ضمن الفترة [1…n]

1. الحالة الأفضل :

$$T^{B}_{SeqSearch}(n) = \Theta(1)$$

2. الحالة الاسوأ :

$$T^{W}_{SeqSearch}(n) = \Theta(1)$$

3. الحالة المتوسطة :

$$T^{A}_{SeqSearch}(n) = \frac{\sum_{i=1}^{n}(n-i+1)}{n}$$

$$= \frac{(n+1)}{2} = \Theta(n)$$

$$\frac{1}{2}(n+1) = \frac{1}{2}n + \frac{1}{2}$$ لأن

أما بالنسبة لحساب تعقيدات البحوث الفاشلة :

$$T_{SeqSearch}(n) = \Theta(n)$$

ملاحظة// يمكن تطبيق أو استخدام خوارزمية البحث التعاقبي للبحث عن عنصر في مصفوفة ثنائية الإبعاد وطبعاً سيكون هناك فرق من حالة إلى أخرى .

التعقيدات العملية (Practical Complexities):

إن تعقيدات الوقت لبرنامج معين تكون وبصورة عامة هي دالة بخصائص المثال وهذه الدالة مفيدة جداً في تحديد كيفية تغير متطلبات الوقت بتغير خصائص المثال ،تستخدم دالة التعقيدات أيضا لمقارنة إيه برنامجين يقومان بانجاز نفس المهمة .

ولنفترض انه لدينا البرنامج P يحوي تعقيدات وقت هي $\frac{P}{\Theta(n)}$ والبرنامج Q يحوي تعقيدات وقت هي $\frac{Q}{\Theta(n^2)}$

فالسؤال هنا هو أي البرنامجين أفضل ؟

ولحل هذا المثال // علينا إتباع التالي

بما إن لكل برنامج حديد احدهما يكون حداً اعلي والأخر يكون ادني هذا يعني إن

زمن تنفيذ البرنامج P الذي حده الأعلى هو αn لقيمة معينة α ولجميع قيم $n \geq n_1$ حيث n تمثل $g(n)$ و α تمثل الثابت C

زمن تنفيذ البرنامج Q الذي حده الأعلى هو βn^2 لقيمة معينة β ولجميع قيم $n \geq n_2$

و حيث $\alpha n \leq \beta n^2$ عندما $n \geq \frac{\alpha}{\beta}$

فان البرنامج P يكون أسرع من البرنامج Q عندما

$$n \geq \max\left(\frac{\alpha}{\beta}, n_1, n_2\right)$$

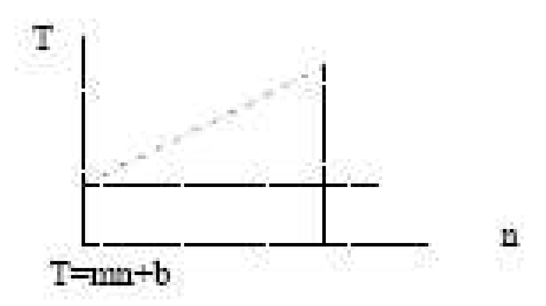
فلو فرضنا إن البرنامج P ينفذ فعلياً في $10^6 n$ ملي ثانية بينما البرنامج Q ينفذ في n^2 وعندما تكون:

$$n \leq 10^6$$

8-1 قياس الانجازية (Performance Measurement) :

إن هذا الموضوع يختص بقياس الوقت على الحاسب والفائدة منه هي تأكيد قياس الوقت على الحاسب،كما نهدف من خلاله للحصول على المتطلبات الحقيقية خزناً ووقتاً للبرنامج (هذه تعتمد على الحاسب وعلى المؤلف أو المترجم والخيارات المستخدمة).
إن وقت تشغيل البرنامج هو ما سنركز عليه في عملنا حيث للحصول على وقت تشغيل لبرنامج معين فانه نحتاج إلى إجراء تجربة وللتخطيط لهذه التجربة يجب اعتبار الجوانب التالية :

1. ماهي دقة الساعة وما هي دقة النتائج التي يرغب بها ،إن لمعلومية دقة النتائج المطلوبة يمكن تحديد طول اقصر حدث يقاس وقته وحيث إن دقة الساعة هي (0.01) من الثانية فيمكن قياس دقة حدث (برنامج) لا يقل عن (1) ثانية للحصول على دقة (0.01).
2. لكل حجم مثال n نحتاج إن نحدد عامل التكرار حيث يختار بحيث يكون وقت الحدث مساوياً الأقل لأقل وقت يمكن قياسه بالدقة المرغوبة .
هناك مبدأ لقياس وقت حدث صغير أو قصير فانه يكون ضرورياً تكراره عدداً من المرات ثم قسمة الوقت الكلي على عدد مرات تكراره.
3. هل سنقيس انجازية أسوأ حالة أم الحالة المتوسطة ، إن بيانات الاختبار تولد تبعاً لحالة ولا توجد إستراتيجية أو سياسة ثابتة حيث يتم الاعتماد على الخوارزمية وغالباً ما يتم اعتماداً على إعداد عشوائية .
4. ما غرض التجربة هل هي لمقارنة خوارزميات أم للتنبؤ بوقت الخوارزمية ،حيث لعملية التنبؤ نحتاج إلى طرح وقت توليد البيانات ودورة التكرار الخاصة بإطالة فترة الحدث (تكرار التجربة).
أما بالنسبة لحالة المقارنة فلا نحتاج لطرح أي شيء طالما كانت ثابتة في جميع الحالات ، أما في حالة التنبؤ الخطي فإننا نحتاج إلى معرفة التعقيدات التقاربية لبرنامج معين فإذا كانت خطية نفصل خطأ مستقيماً بأقل مربعات انحراف ،فإذا كانت النتائج المحسوبة نظرياً خطية فإنها في الجانب العملي أو التطبيقي ستكون شبيهة بالتربيعية وهناك ثوابت تهمل بالنظري وتحسب بالجانب العملي فلو فرضنا العلاقة التالية:



شكل(5):حالة التنبؤ في تحديد الانجازية

حيث إن :
T : يمثل الوقت
m :يمثل الميل
n :يمثل عدد العناصر
b :يمثل قاطع المستقيم مع الاحداثي y
فإذا كانت خطية فالعلاقة ستكون هي :

$$m = \frac{t - b}{n}$$

وفي حالة أنها أصبحت تربيعية فهنا تفصل قطع مكافئ

$$t = a_0 + a_1 n + a_2 n^2$$

أما إذا كانت التعقيدات هي $\Theta(n.m)$ فانه تفصل منحنياً ذو صيغة

$$t = a_0 + a_1 n + a_a nLogn$$

مثال// افترض قياس انجازية أسوا حالة لخوارزمية البحث التعاقبي (Sequential Search) ؟

الاستنتاج //

إن الهدف من التجربة هو قياس الحالة الأسوأ ،حيث إن هذه الخوارزمية تحتاج وقت قليل جداً واقل وقت في الحاسبات هو اقل من جزء من الثانية ،لذلك تحتاج إن نعمل دوارة لزيادة الوقت كما في جزء البرنامج التالي :

```
# include <iostream.h>
# include <iomanip.h>
# include <time.h>
Void time Search( )
{ // repetition factors
  Long int r[21]={0,200.000,200.000,1500.000,……25000}
  Int a[1001],n[21];
For (int j=1;j<=1000,j++)  a[j]=j;
For (int j=1;j<=10,j++)
{ n[j]=10*(j-1);
   n[j+10]=100*j;
 }
Cout<<" n     t1       t "<<endl<<endl;
Cout<<Setprecision(6);
For (int j=1;j<=20,j++)
{ int h=Gettime( );
For (int i=1;i<=r[j],i++)
{ k=SeqSearch(a,0,n[j]);
  Int h1=Gettime( );
  Int t1=h1-h;
  Float t =t1 ;
  t/=r[j];
  cout<< Setw(5)<<n[j]<< Setw(5)<<t1<< Setw(8)<<t<<endl;
}
  cout <<" time are in millisecond..<<endl;
}
```

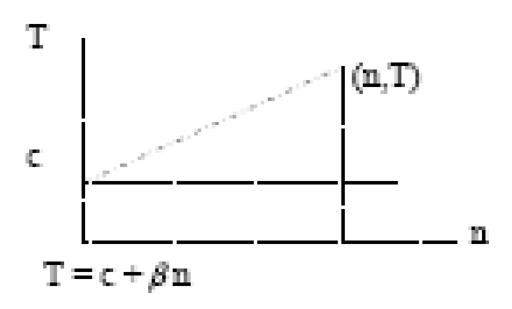
وفيما يلي توضيح موجز للبرنامج:

- المصفوفة [21]r تستخدم لخزن قيم التكرارات التي تبدأ من 1 إلى 20 قيمة أي إننا سوف نأخذ 20 قيمة للـ n وهي غير ثابتة. حيث إن الغاية من هذه التكرارات هي جعل الوقت اقل من الثانية ، مع ملاحظة انه كلما زادت n فان عامل الوقت يقل وبالتالي يجب تقليل التكرارات .
- إن لكل قيمة في مصفوفة الـ n تكرار يقابله في مصفوفة الـ r ، وحجم مصفوفة الـ n هو 21 حيث إننا نضحي بالعنصر الذي نبحث عنه والذي يملك الموقع رقم 1 بالمصفوفة ،أما مصفوفة a فهي تحوي العناصر التي نبحث فيما بينها عن العنصر المطلوب .
- إن المعادلة التي تخص ملئ مصفوفة n بالقيم هي معالجة ليست ثابتة ويمكن تغييرها من مسألة إلى أخرى.
- إن الدالة Gettime() هي إجراء افتراضي يعيد الوقت الحالي للمتغير h مقاساً بأجزاء مئوية من الثانية فإذا قلنا إن القيمة هي 10 هذا يعني ملي ثانية وإذا كانت القيمة 100 فانه يعني ثانية.
- للمتغير k الذي خصصنا له نتيجة دالة البحث فان القيمة 0 الموجودة ضمن معطيات الدالة تعني أي عنصر غير موجود بالمصفوفة a ،وان قيمة تختلف عن قيمة وتحوي الفرق في الوقت.
- الدالة Setprecitsion(6) تكون خاصة بالمراتب ما بعد الفارزة للإعداد العشرية ،أما دالة Setw(5) فهي تعني الانتقال مسافة بحجم الرقم المحدد ضمن السطر الحالي .

إن نتيجة هذا البرنامج هي الحصول على وقت الدالة (Sequential Search) مضافاً إلية وقت دوارة التكرارات وللتخلص من وقت دوارة التكرارات علينا إن نعيد نفس التجربة بالإضافة إلى حذف جسم الدوارة أي إننا نجعل جسم الدوارة فارغاً (for(;;)) ونسجل الزمن الذي تلخذة كل دورة ثم نطرحه من قيم الزمن t ، في التجربة السابقة (قيمة وقت الدورة الواحدة) تطرح من كل قيم T .

في التجربة السابقة كان وقت الدورة الواحدة تقريباً (0.002) ولكنة غير ثابت أي انه قد يكون اقل من ذلك بكثير في الحاسبات ت ذات السرعة العالية.

وهناك علاقة تربط قيم التكرارات (n) بالزمن (T) وهي علاقة الزمن بالحجم في الدالة Sequential Search



شكل(6): علاقة الزمن بالحجم في الدالة Sequential Search

الفصل الثاني

الترتيب

(Sorting)

1-2: خوارزميات الترتيب (Sorting Algorithms):

الخوارزمية هي عبارة عن مجموعة من الخطوات المتسلسلة و الرياضية والمنطقية اللازمة لحل مشكلة ما ، و سميت الخوارزمية بهذا الاسم نسبة إلى العالم المسلم" أبو جعفر محمد بن موسى الخوارزمي ".

خوارزمية الترتيب هي خوارزمية تُمكن من تنظيم مجموعة عناصر حسب ترتيب محدد، العناصر المراد ترتيبها توجد في مجموعة مزودة بعلاقة ترتيب معينة.

تصنيف خوارزميات الترتيب مهم جدا، لأنه يُمكن من اختيار نوع الخوارزمية الأكثر مناسبة للشكل المعالج، مع الأخذ بعين الاعتبار السلبيات الموجودة في الخوارزمية.

بمعنى أخر الترتيب عبارة عن عملية ترتيب مجموعة من العناصر البيانية وفق قيمه معينة تسمى حقل أو وفق حقول تسمى المفتاح أما بصورة تصاعدية أو تنازلية .
الغرض من الترتيب هو :
1 - زيادة كفاءة الخوارزمية " البحث عن عناصرها ".

مثال// تمثيل الأرقام 3 و4

3	4	النظام العشري
11	100	النظام الثنائي

0	0	0	0	0	1	1

خسارة في المساحة الخزنية لأن الاصفار تشغل موقع

0	0	0	0	1	0	0

استخدام 2 بايت للخزن

0	0	1	0	0	1	1

استخدام بايت 1

2 - تبسيط معالجة الملفات :
لأن الملفات تتألف من حقول فإن ترتيب هذه الملفات حسب مفاتيح يكون أسهل في عملية البرمجة والبحث .
مثال// ملف يتألف من ثلاث حقول أول حقل هو تسلسل الطالب والثاني أسمة والثالث المعدل ، نستطيع أن نستخدم مفتاح معين لترتيب الأسماء حسب التسلسل والمعدل .

3 - حل مشكلة تشابه القيود :
توجد مشكلة في القيود هي تشابه الأسماء فإذا كان الاسم مشابه لاسم أخر نأخذ اسم الأب وإذا كان اسم متشابه نأخذ اسم الجد ، مثل اسم زينب
زينب حيدر محمد
زينب حيدر حسن

2-2: أنواع الترتيب (Types of Sorting) :

1- الترتيب الداخلي(Internal Sort)
2- الترتيب الخارجي (External Sort)
* الترتيب الداخلي : يحدث داخل الذاكرة بحيث يكون حجم البيانات مناسب وليس كبير ،ويشمل
1- ترتيب الاختيار(Selection Sort)
2- ترتيب الفقاعي (Bubble Sort)
3- ترتيب الإضافة (Insertion Sort)
4- ترتيب شيل (Shell Sort)
5- الترتيب السريع (Quick Sort)
6- ترتيب الأساس (Radix Sort)
7- ترتيب المؤشرات (Pointers Sort)
8- الترتيب الشجري لشجرة البحث الثنائية (Tree Sort)
9- Topological sorting

* الترتيب الخارجي : هو الترتيب الذي يحدث خارج الذاكرة في أوساط الخزن الثانوي عندما يكون حجم البيانات كبير بحيث يتعذر استيعابها في الذاكرة أثناء عملية الترتيب ويشمل
1- الترتيب بالدمج (Merge Sort)
2- الترتيب بالدمج المتوازن ذو المسارين (Balanced Two Way Merge Sort)
3- الترتيب بالدمج باستخدام طريقة قسم وانتصر (Divided & Conquer Mirage Sort) .

العوامل الرئيسية المحددة لاختيار خوارزمية الترتيب :

1- حجم البيانات المخزونة : أذا كان صغير يكون خزن داخلي أما أذا كان كبير يكون الخزن الخارجي .
2- نوع الخزن: أذا كان ذاكرة رئيسية يكون الخزن داخلي أما أذا أشرطة مغناطيسية يكون الخزن خارجي.
3- درجة ترتيب البيانات: حيث إن البيانات الشبة مرتبة تترتب بشكل أسرع من البيانات غير المترتبة أطلاقا.

3-2: خوارزميات الترتيب الداخلي (Internal Sort):

1- خوارزمية الاختيار (Selection Algorithm)

ترتيب الاختيار هو خوارزمية الترتيب الأكثر بديهية ، و يتم عن طريق البحث إما عن العنصر الأكبر أو عن العنصر الأصغر و الذي يوضع في المكان الأخير، ثم نبحث عن ثاني أكبر أو أصغر عنصر و الذي يوضع في مكانه أي قبل المكان الأخير، إلى آخره... حتى يتم ترتيب الجدول بأكمله.

خصائص ترتيب جدول ما :

1. عدد المقارنات اللازمة لترتيب جدول عدد عناصره N هو N(N-1)/2 .
2. عدد التبديلات في رتبة N .

ويمكن توضيح ذلك حسب الخطوات الآتية:
1- أيجاد أصغر عنصر في القائمة واستبداله من موقعة مع العنصر في الموقع الأول في القائمة.
2- أيجاد أصغر عنصر من المتبقي في القائمة واستبداله من موقعة مع الموقع الثاني في القائمة.
3- تستمر هذه العملية حتى الوصول إلى العنصر الأول .

مثال// رتب العناصر الآتية باستخدام طريقة الإختيار (Selection Algorithm).
8 , 3 , 9 , 7 , 2 , 6 , 4

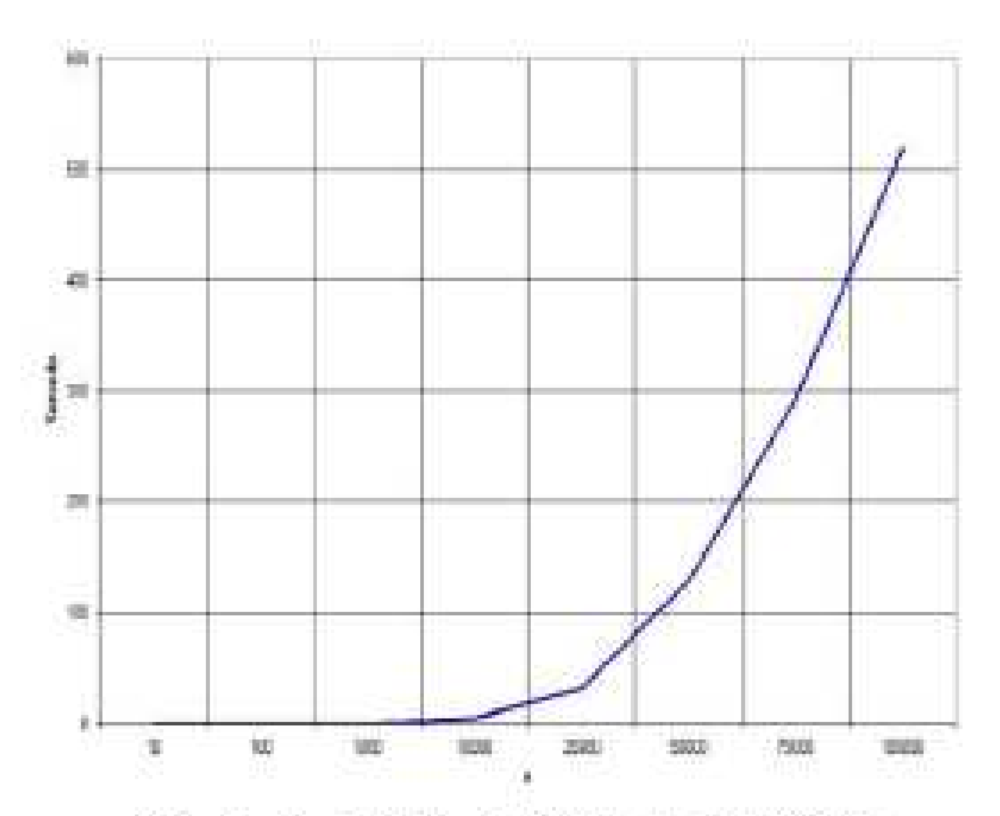
List	1	2	3	4	5	6
8	2	2	2	2	2	2
3	3	3	3	3	3	3
9	9	9	4	4	4	4
7	7	7	7	6	6	6
2	8	8	8	8	7	7
6	6	6	6	7	8	8
4	4	4	9	9	9	9

الإستنتاج:
عدد العناصر N=7
عدد المراحل N-1 = 6
ملاحظة // معدل المقارنات هو : N/2 * (N-1) حيث
برهن ذلك ؟ 6*3.5 =21
نستنتج أن عدد المقارنات يعتمد على عدد المراحل ويتناقص في كل مرحلة بواحد إلى أن يصل إلى الواحد ، إن عدد المراحل التي يترتب بها هي N-1 = 6 ومعدل التبديلات هو 21 ، المشكلة في طريقة الإختيار التي لاحظناها في هذه الخوارزمية هي أن كل عنصر يمكن أن يقارن في كل مرة بقيمتين لانه يمكن أن يبدل موقعة .

مثال// لديك قائمة فيها ثلاث عناصر مثلا العنصر الذي قيمته (3) كان في الموقع (2) وأصبح في الموقع (1) أما العنصر (8) في الموقع الثاني بقى في نفس الموقع وبالرغم من ذلك احتجنا إلى مرحلة أضافية لأنه يجب أن يقارن مع الرقم (9) .

8	3	3
3	8	8
9	9	9

التحليل التجريبي (Empirical Analysis) :



شكل(7):فعالية خوارزمية الاختيار (Selection Sort Efficiency).

والدالة البرمجية التي تقوم بتطبيق هذه الطريقة هي:

```
void selectionSort(int numbers[], int array_size)
{
  int i, j;
  int min, temp;
  for (i = 0; i < array_size-1; i++)
  {min = i;
    for (j = i+1; j < array_size; j++)
    { if (numbers[j] < numbers[min])
        min = j;
    }
    temp = numbers[i];
    numbers[i] = numbers[min];
    numbers[min] = temp;
  }
}
```

مثال// برنامج يقوم باستخدام طريقة الترتيب بالاختيار لمجموعة كلمات .

```
#include< stdio.h>
main( )
{  Char *z;
   Char *name[ ] ={"ammer","Fatima","omar" ,"ahmed","jamal",
    "saeed","yousef","mariam"};
    Int nmax=8;
    Register int i,j,k;
    For (i=0;i<=nmax-1; ++i)
    {  k=i;
       z=name[i];
       For(j=i+1; j<nmax;++j )
        {If (strcmp(name[j],z)< 0)
          { k=j;
            z=name[j];
          }
       }
      Name[k]=name[i];
      Name [i]=z;
    }
   For(i=0;i<nmax;++i)
     printf("%s\n", name[i]);
 }
```

Ahmed
Ammar
Fatima
jamal
Mariam
Omer
Saeed
yousef

2- خوارزمية الترتيب الفقاعي (Bubble Sort Algorithm):

ترتيب الفقاعات خوارزمية ترتيب منتقدة لبطئها، وهي تعمل على رفع العنصر الأكبر كفقاعة الهواء التي ترتفع إلى أعلى وذلك بترتيب العناصر بتتابع، أي تقوم بمقارنة العنصرين الأول و الثاني، تحتفظ بالعنصر الأكبر، و تبدل الأماكن إذا كانا غير مرتبين. تقوم بهذه العملية إلى آخر عنصر، بعد ذلك تعيد العمليات إلى المكان ما قبل الأخير وهكذا دواليك.. تتوقف عند وجود جدول بالبعد 1 أو عندما لا تقوم بالتبديلات عند آخر عملية.

لترتيب جدول A بعده N،فان عدد المقارنات سيكون: $\frac{N(N-1)}{2}$

أما عدد التبديلات فهو في المتوسط : $\frac{N(N-1)}{4}$

تقوم هذه الخوارزمية بترتيب مجموعة أعداد n ترتيباً تصاعدياً على عدة مراحل عددها n-1 بحيث يتم وضع عدد واحد على الأقل في ترتيبه الصحيح بنهاية كل مرحلة.

خطوات تطبيق الخوارزمية :

1- إدخال الأعداد المراد ترتيبها في مصفوفة X .
2- استخدام متحول switched تدل قيمته على حدوث (switched=FALSE) أو عدم حدوث تبديل (switched=TRUE) .
3- استخدام حلقة خارجية بعدد المراحل أي n-1 بحيث تتوقف الحلقة في حال عدم حدوث تبديل (الأعداد مرتبة).
4- استخدام حلقة داخلية لمقارنة كل عدد بالعدد الذي يليه، حيث يتم تغيير قيمة المتحول switchedإلى true في حال التبديل.
5- استخدام حلقة داخلية لإظهار ترتيب الأعداد في نهاية كل مرحلة.
6- استخدام حلقة لإظهار ترتيب الأعداد النهائي.

وهذا جزء البرنامج الخاص بتطبيق الخوارزمية:

```
#include <iostream.h>
#define MAXNUM 20
enum boolean{FALSE,TRUE};
void main()
 {int X[MAXNUM];
  int n,i,pass,hold;
  int switched=TRUE;
  cout<<"Enter count of numbers\t";
  cin>>n;
  for(i=0;i<n;i++)
     cin>>X[i];
  for(pass=0;pass<n-1 && switched==TRUE; pass++)
  {  switched=FALSE;
     for(i=0;i<n-1-pass;i++)
      if(X[i]>X[i+1])
       { switched=TRUE;
         hold=X[i];
         X[i]=X[i+1];
         X[i+1]=hold;
       }
```

```
      for(i=0;i<n;i++)
        cout<<X[i]<<"\t";
        cout<<endl;
    }
  cout<<"The sort is :\n";
  for(i=0;i<n;i++)
  cout<<X[i]<<"\t";
}
```

أن فكرة هذه الطريقة تتضمن أيجاد أصغر القيم ووضعه في قمة القائمة،حيث تقسم إلى مرحلتين هما :

1- First pass
2- Second pass

حيث نبدل موقعهما ليكون الأصغر أعلى القائمة لحين الوصول N ,N-1 وكما يلي:

1- نقارن العنصرين في الموقعين إلى العنصر في الموقع الثاني لأن الموقع الأول قد اختير سابقا
2- نقارن بنفس الطريقة من العنصر في الموقع N
3- نكرر الخطوات لـN-1 من المراحل.

مثال// ترتب العناصر الآتية بطريقة الترتيب الفقاعي (Bubble Sort Algorithm).

8 , 3 , 9 , 7 , 2

List	pass(1)	pass (2)	pass (3)	pass (4)
8	8 8 8 2	2 2 2	2 2	2
3	3 3 2 8	8 8 3	3 3	3
9	9 2 3 3	3 3 8	8 7	7
7	2 9 9 9	7 7 7	7 8	8
2	7 7 7 7	9 9 9	9 9	9

عدد العناصر $N = 5$

عدد المراحل $N\text{-}1 = 4$

عدد المقارنات $N^2/2=25/2=12.5$

معدل عدد التبديلات $N^2/4=25/4= 6.25$

أن هذه الطريقة تكون جيدة أذا كانت العناصر شبه مرتبه و عددها ليس كبير فلا تحتاج إلى مساحة خزنيه كبيرة لهذا فأن وقت التنفيذ لهذه الخوارزمية $O(N^2)$

3- خوارزمية الإضافة (Inserting Sort Algorithm):

تتلخص هذه الخوارزمية كما يلي :-

1- نبدأ بالعنصر 2 في القائمة ونقارنه مع العنصر الأول ونضعه حسب الترتيب في مقدمة القائمة ولكن ترتيب تصاعدي.

2- نبدأ بالعنصر 3 ونقارنه مع مقدمة القائمة التي تحتوي على العنصر الأول والثاني ونضعه في الموقع الثاني ونستمر بالعملية لحين الحصول على قائمة مرتبة.

مثال// رتب العناصر الآتية بطريقة ترتيب الإضافة (Inserting Sort Algorithm)

8, 3 , 9 , 7 , 2 , 6 , 4

الحل// يمكن ترتيبها كما في الجدول التالي :

List	1	2	3	4	5	6
8	3	3	3	2	2	2
3	8	8	7	3	3	3
9	9	9	8	7	6	4
7	7	7	9	8	7	6
2	2	2	2	9	8	7
8	6	6	6	6	6	9
4	4	4	4	4	4	9

إن ترتيب الإضافة هو عكس ترتيب الاختيار لأنه يأخذ العنصر ويقارنه مع العنصر الذي قبلة . حيث تقارن الأول مع الثاني والأول مع الثالث هكذا .

عدد العناصر $N= 7$

عدد المراحل هو $N-1 =6$

معدل عدد المقارنات هو $N^2/4$

معدل التبديلات هو $N^4/4$

مثال// برنامج يقوم بتطبيق خوارزمية ترتيب الإضافة .

```
typedef int tab_entiers[MAX];

void insertion(tab_entiers t) {
   /* Specifications externs */
      int i,p,j,x;
      for(i = 1 ; i < MAX ; i++)
      {     /* position dissertation */
            /* determine  p       0 <= p <= i */
            /* t[p] >= t[i] */
            p = 0;
            while(t[p] < t[i]) p++;
            x = t[i];       /* t[i] */
            For (j = i-1 ; j >= p  ; j--)  t[j+1] = t[j];
             /* translation t[p..i-1] vers t[p+1..i] */
            t[p] = x; /* insertion  t[p] */
       }
}
```

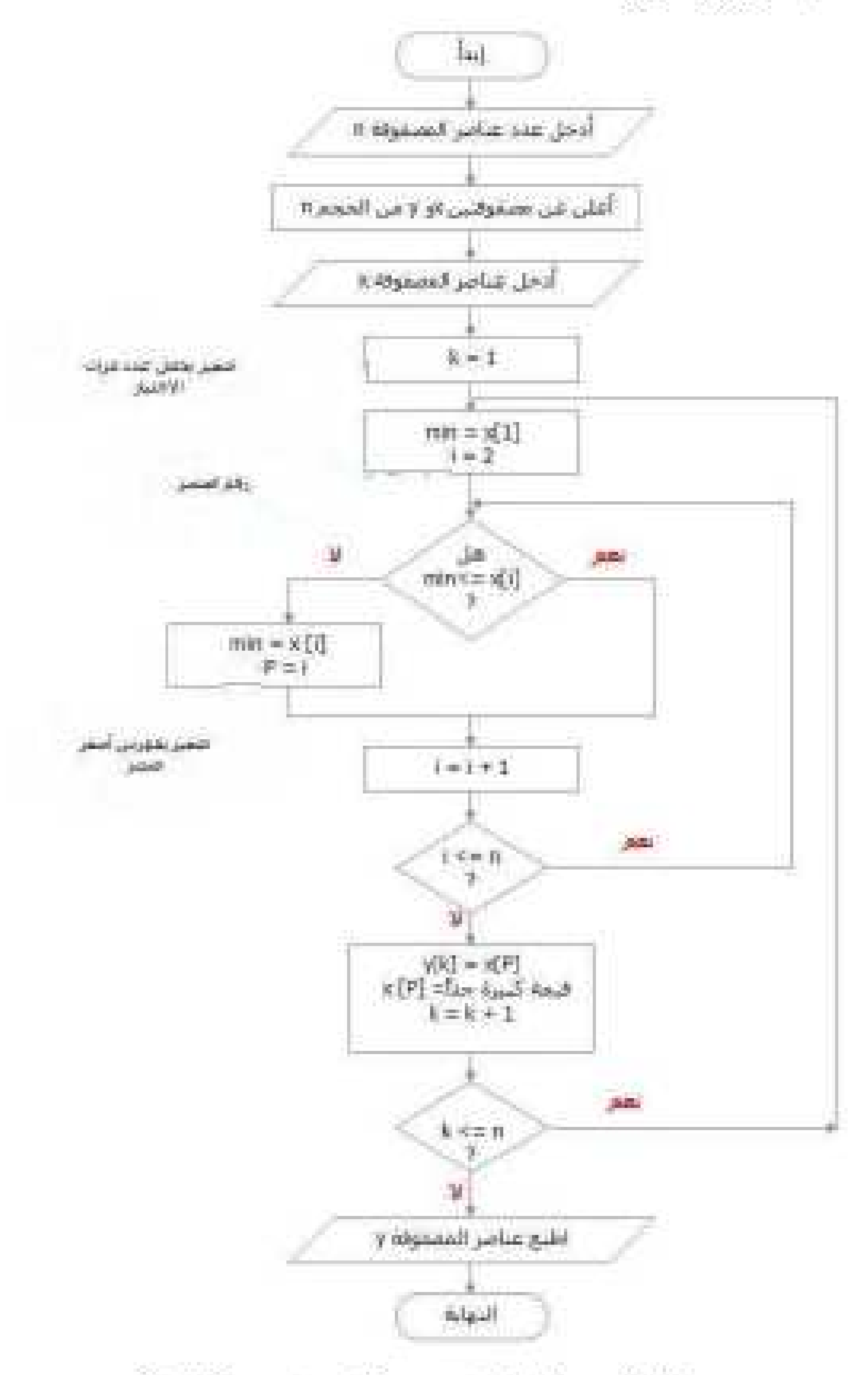
مثال// قائمة تحتوي مجموعة عناصر ، العناصر المظللة بالرمادي هي العناصر المختارة أو المحددة والتي يراد ترتيبها، بينما العناصر المكتوبة بالخط العريض bold هي العناصر المرتبة في مكانها الصحيح:

Initial Array	29	10	14	37	13

الحل// يمكن توضيحه بالخطوات المبينة ادناه :

After 1st swap:	29	10	14	13	**37**
After 2nd swap:	13	10	14	**29**	**37**
After 3rd swap:	13	10	**14**	**29**	**37**
After 4th swap:	**10**	**13**	**14**	**29**	**37**

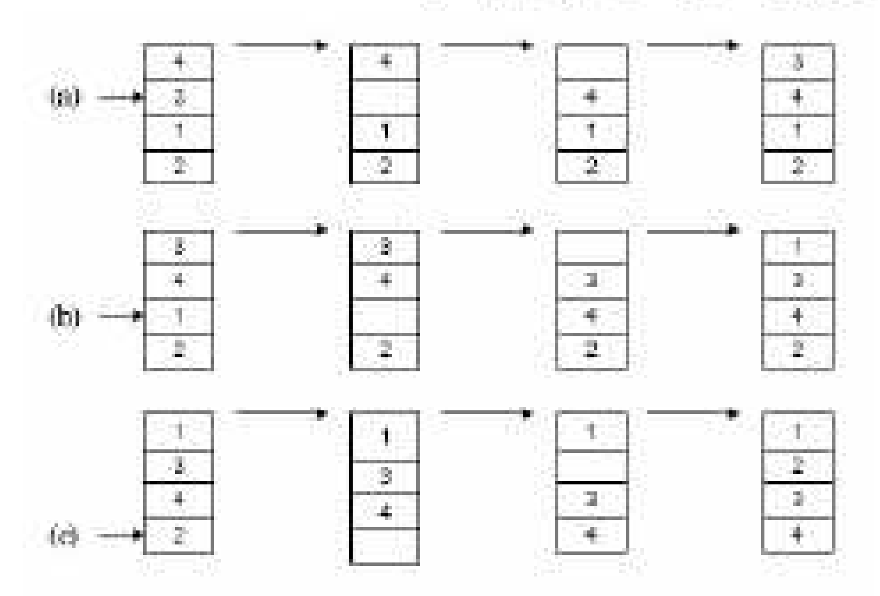
من الممكن أن نستخدم مصفوفتين لعمل ذلك، بحيث تحوي الأولى العناصر غير المرتبة ويتم تخزين هذه العناصر بترتيب تصاعدي أو تنازلي في المصفوفة الثانية، ويمكن كتابة flowchart باستخدام مصفوفتين كما يلي:



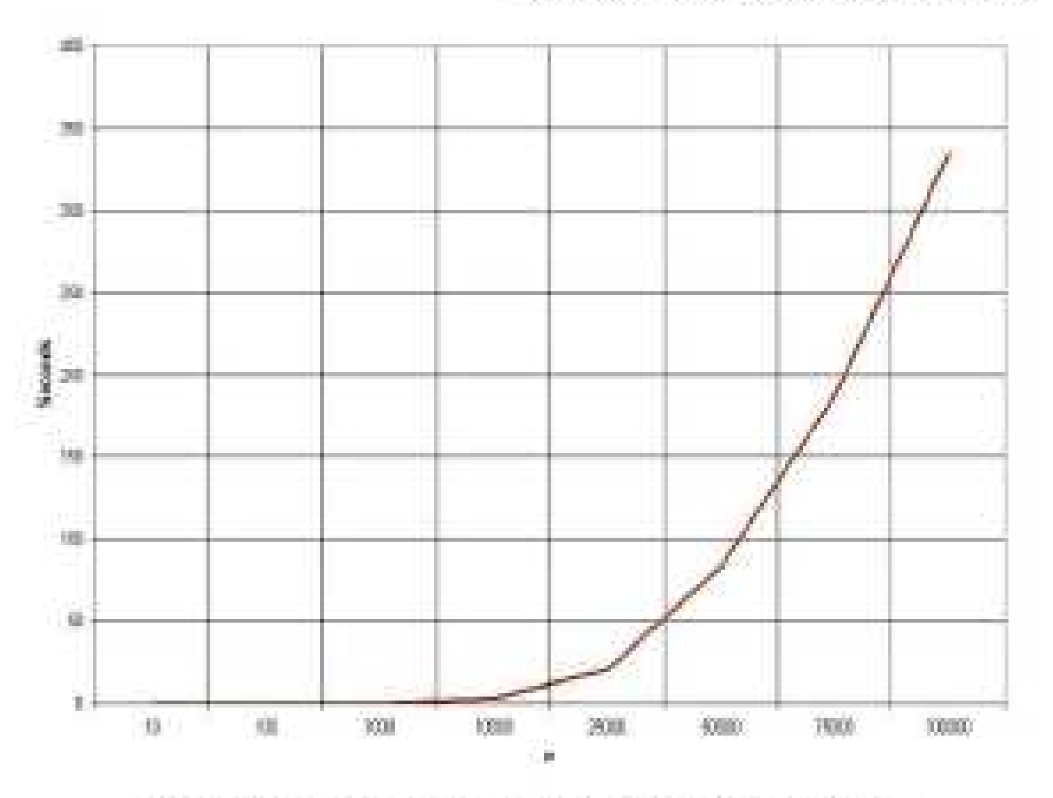
شكل(8): مخطط انسيابي يوضح فكرة خوارزمية الإضافة

مثال// رتب عناصر القائمة الآتية (4,3,1,2) باستخدام خوارزمية الإضافة .

الحل// يمكن ترتيبها كما في الخطوات (a,b,c) التالية :



التحليل التجريبي (Empirical Analysis) :



شكل(9):فعالية خوارزمية الإضافة (Insertion Sort Efficiency)

والجزء البرمجي الخاص بتطبيق خوارزمية الإضافة هو :

```
void insertionSort(int numbers[], int array_size)
{
  int i, j, index;

  for (i=1; i < array_size; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0) && (numbers[j-1] > index))
    {
      numbers[j] = numbers[j-1];
      j = j - 1;
    }
    numbers[j] = index;
  }
}
```

مثال// برنامج يقوم باستخدام خوارزمية ترتيب الإضافة لترتيب مجموعة كلمات

```
#include< iostream.h>
main( )
  {
   Char *z;
   Char *name[ ] ={"ammer","Fatima","omar" ,"ahmed","jamal",
                        "saeed","yousef","mariam"};
    Int nmax=8;
    Register int i,j;
    For (i=0;i<=nmax-1; ++i){
        z=name[i];
        j=i-1;
        While (j>=0 &&(strcmp(z,name[j]<0)){
          Name[j+1]=name[j];
           J--;
         }
        name[j+1]=z;
        }

        For(i=0; i<nmax;++i )cout<<"/n"<< name[i];
}
```

Ahmed
Ammar
Fatima
jamal
Mariam
Omer
Saeed
yousef

3- خوارزمية شيل (Shell Sort Algorithm):

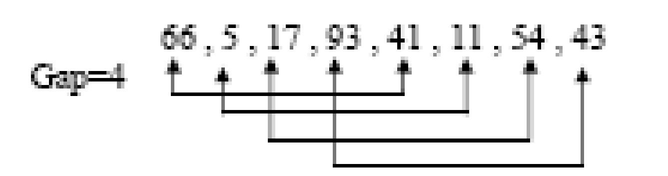
توجد مشكلة في الترتيب الفقاعي هي (أن عدد المقارنات تزداد لكل عنصر أذا زادت عدد العناصر أو الأعداد في القائمة) مثلا أذا كان العنصر في أخر ترتيب (في أخر تسلسل في القائمة) وأن موقعه الصحيح يجب أن يكون في الموقع الأول ، نحتاج هنا إلى عدد كبير من المقارنات وهذا يؤدي إلى كثرة الأخطاء .

والحل لهذه المشكلة من خلال خوارزميتين هما :
1- خوارزمية شيل
2- خوارزمية الترتيب السريع
فكرتها تتلخص كالآتي : حيث نقوم بتقسيم القائمة إلى مسافات وهمية ، وتجري مقارنة بين عنصرين أو أكثر ليس متجاورين وإنما متباعدين بالمسافة المحددة ، ثم تختصر المسافة الوهمية إلى النصف وتجري المقارنة أو التبديل ثانية إلى أن تصبح المسافة تساوي (1)، وبذلك يتم ترتيب القائمة .
إن المسافة الوهمية بين عنصرين تدعى فجوة (Gap) .

مثال// لديك العناصر التالية 66,5,17,93,41,11,54,43

الحل //
1 - نحسب الأعداد المراد ترتيبها N=8
2 - نقسم المسافة الوهمية إلى النصف Gap = 4

المرحلة الأولى: 66 , 5 , 17 , 93 , 41 , 11 , 54 , 43



41 , 5 , 17 , 43 , 66 , 11 , 54 , 93

المرحلة الثانية: Gap =4/2 = 2

41 , 5 , 17 , 43 , 66 , 11 , 54 , 93

17 , 5 , 41 ,43 , 66, 11, 54, 93

17 , 5 , 41 ,11 , 66,43 , 54 , 93

17 , 5 , 41 ,11 , 54 ,43 , 66, 93

المرحلة الثالثة: Gap = 2/2 = 1
في هذه المرحلة نستمر بعملية التبديل حتى نحصل على القائمة مرتبة الآتية:

5 , 11 , 17 , 41 , 43 , 54 , 66 , 93

خصائص تطبيق خوارزمية شيل (Shell Sort Algorithm):

1 - تزداد كفاءتها كلما زادت عدد القيود
2 - لا تحتاج إلى مكان أضافي في الذاكرة لأجراء عملية الترتيب .
3 - كفوءة أذا كانت القيود داخل القائمة مرتبة أو شبة مرتبة
والسبب لكل فقرة أعلاه هو :
1 - تزداد كفاءتها لأنه يتم ترتيب القائمة قبل الوصول إلى Gap≠1
2 - وذلك لإنه يمكن أن يكون تبديلين أو أكثر للعنصر الواحد بدون استخدام مكان خاص له
3 - وذلك لإنه لا توجد هنا تبديلات بالأرقام أذا كانت مرتبة أو عدد التبديلات أقل أذا كانت الأرقام شبة مرتبه .

قوانين خاصة بتطبيق خوارزميه شيل (Shell Sort Algorithm):

تستخدم هذه القوانين للحصول على الحالة الإكفأ للترتيب ،وكذلك في حالة القوائم الكبيرة .

القانون الأول: اختيار أفضل Gap حيث :$Gap= 1.72*(N^1/3)$

في المثال السابق استخدمنا : N = 8
=4.586666667≈5

القانون الثاني: اختيار أفضل قيمة لمعدل الوقت (Time average) حيث :
$Tav = N^(5/3)$
$= 8^{(5/3)} =32$

مثال// رتب عناصر القائمة الآتية (3,5,1,2,4) بطريقة شيل مستخدماً فجوة مقدارها (2) مرة واخرى (1) ؟

الحل// يمكن ترتيبها كما في الخطوات الآتية:

(a)- Gap=2 3, 5, 1, 2, 4

1, 2, 3, 5, 4

نقسم الفجوة فتكون: Gap=1 1, 2, 3, 5, 4

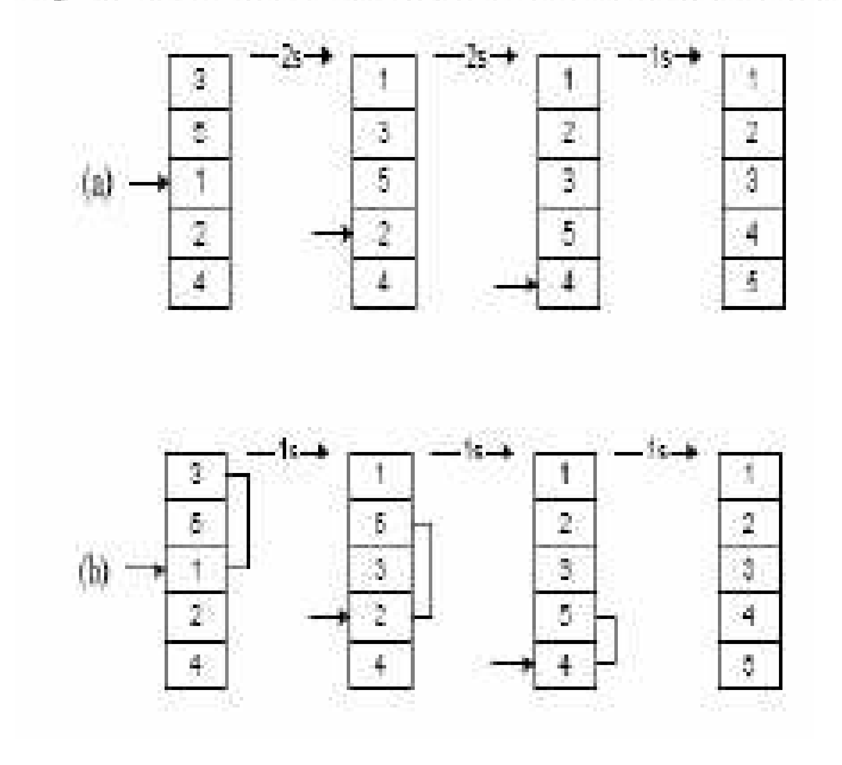
1, 2, 3, 4, 5

(b) - Gap=1 3, 5, 1, 2, 4

نستمر بعملية التبديل حتى نحصل على قائمة مرتبة:

1, 2, 3, 4, 5

ويمكن تمثيل عملية التبديل ما بين عناصر القائمة بالنسبة للعنصر الأصغر كما يلي :

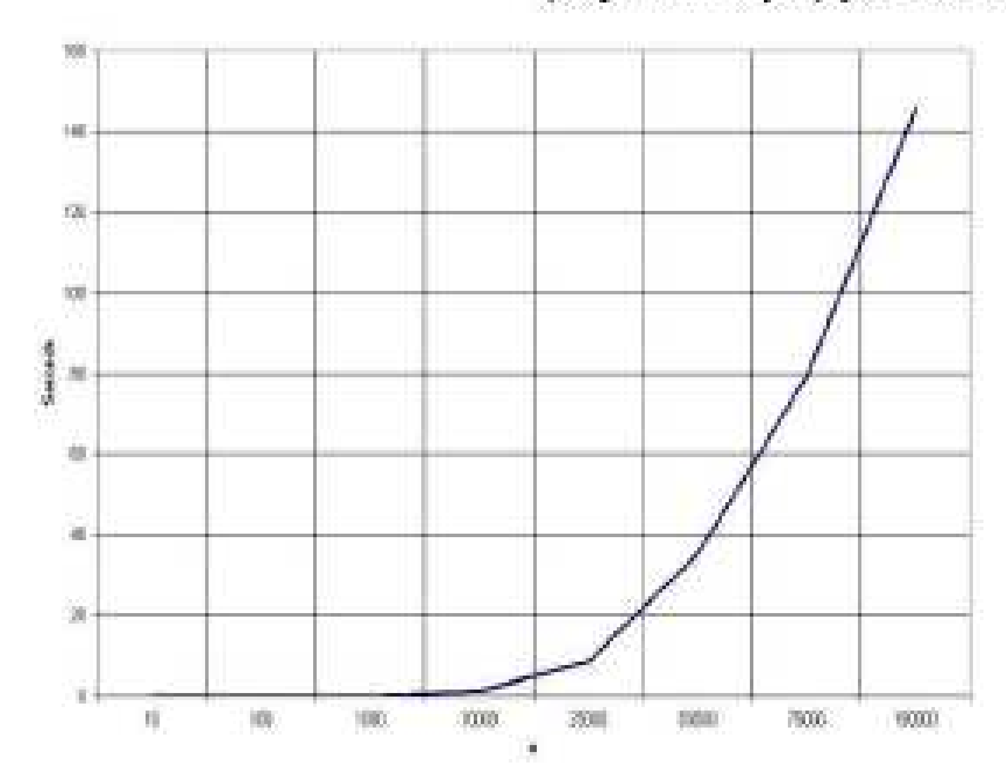


مثال//البرنامج التالي يستخدم خوارزمية ترتيب شيل (shell) لترتيب مجموعة كلمات .

```
#include< stdio.h>
main( )
 {
   Char *name[ ] ={"ammer","Fatima","omar" ,"ahmed","jamal",
                         "saeed","yousef","mariam"};
    Int m[ ]={9,5,3,2,1};
    Int nmax=8;
    Register int a,b,c,t,v;
    Char *z;
    For (v=0;i<=5; ++v){
          c=m[v];
           t=-c;
           for  (a=c;a<nmax;++a){
            z =name[a];
            b=a-c;          }
        If(t==0){
        t=-c;
        t++;
        name[t]=z;
   }
   While((strcmp(z,name[b]))<0)&&(b>> 0)&&(b<nmax)){
      Name[b+c]=name[b];
       b-=c;
     }
      Name[b+c]=z;
     }
      }
       For(a=0; a<nmax;++a )printf<<"%s\n"<< name[a];
}
```

Ahmed
Ammar
Fatima
jamal
Mariam
Omer
Saeed
yousef

التحليل التجريبي (Empirical Analysis):



شكل(10):فعالية خوارزمية شيل(Shell Sort Efficiency)

والجزء البرمجي الخاص بتطبيق خوارزمية شيل هو:

```
void ShellSort(int numbers[], int array_size)
 {int i, j, increment, temp;
  increment = 3; /* increment=Gap*/
  while (increment > 0)
  {
   for (i=0; i < array_size; i++)
   {
    j = i;
    temp = numbers[i];
    while ((j >= increment) && (numbers[j-increment] > temp))
    {
     numbers[j] = numbers[j - increment];
     j = j - increment;
    }
    numbers[j] = temp;
   }
   if (increment/2 != 0)
    increment = increment/2;
```

```
    else if (increment == 1)
      increment = 0;
    else
      increment = 1;
  }

}
```

5- خوارزمية الترتيب السريع (Quick Sort Algorithm) :

الترتيب السريع هو طريقة ترتيب من اختراع هوار (C.A.R.Hoare) في 1962.

خصائص الخوارزمية:

تعتمد الخوارزمية في عملها على وضع العنصر الأول (يسمى مؤشر) في مكانه النهائي ثم وضع العناصر الأكبر من المؤشر من جهة اليمين و العناصر الأصغر من جهة اليسار، و تسمى هذه العملية تجزئة، ثم نقوم بإجراء عملية تجزئة بالنسبة لكل جهة (اليمين ، اليسار)، حيث نحدد مؤشرا جديدا و نعيد عملية التجزئة،تتكرر هذه العملية إلى أن نحصل على مجموعة مرتبة.

إذا تم اختيار المؤشر بطريقة صحيحة، نحصل على الطريقة الأسرع للترتيب في الحالة المتوسطة، مع تعقيد بـ ((O(n Ln (n والتي قد تتحول إلى (($O(n^2)$) في الحالة الأصعب، و هي حالة جدول عناصره مرتبة أصلا، و لكن هذه الحالة بديهية لأن المجموعة مرتبة أصلا.

من الناحية العملية، بالنسبة للتجزئة مع عدد قليل لا يتجاوز بضع عشرات من العناصر، يتم اللجوء عادة إلى الترتيب بالإدراج الذي يكون أفضل من الترتيب السريع.

و بصورة عامة يعتبر الترتيب السريع الأكثر شيوعا (شعبية) من بين جميع خوارزميات الترتيب،حيث المشكلة الوحيدة تتمثل في كيفية اختيار المؤشر.

اختيار أفضل مؤشر:

عند استعمال الترتيب السريع لمجموعة مرتبة مسبقا، و بطريقة اعتباطية، يستغرق كما قلنا وقتا كبيرا، و ذلك بسبب أن أول عنصر هو الذي يعتبر مؤشرا، الشيء الذي يؤدي إلى عدم تقسيم المجموعة إلى قسمين أكبر و أصغر من المؤشر. لحل المشكلة يتم اختيار العنصر الأوسط ، كما يمكن اختياره عشوائيا من عنصرين متواجدين حول المركز.

تكون فكرة خوارزمية الترتيب باستخدام مبدأ التجزئة حيث نقوم بعمل الخطوات الآتية :

1- نقسم القائمة إلى جزئين حيث نختار أحد عناصر القائمة وليكن في الوسط تقريبا نسميه (X).
2- نقوم بعملية المسح باتجاهين بحيث تكون العناصر على جهة اليسار هي الأصغر من (X) أي إننا نقوم بالتبديل ،أما العناصر الموجودة على جهة اليمين فهي الأكبر من قيمة (X) .

3- نأخذ النصف الأوّل ونجري علية عملية ترتيب سريع مرة أخرى كذلك ثانية للنصف الثاني وهكذا إلى أن تكون جميع العناصر مرتبة .

(نصف القائمة الأيمن والأكبر) (X) (نصف القائمة الأيسر والاصغر)

ملاحظه :
* أذا كانت قيمة (N) هي عدد زوجي فان قيمة (X) تكون :

مثلاً N=8
$8 \div 2 = 4$

نمثلها كما الشكل: 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8

* أذا كانت قيمة (N) هي عدد فردي فان قيمة (X) تكون :

مثلاً N=11
$11 \div 2 = 5.5$

5 or 6
نمثلها كما في الشكل : 1,2,3,4,5,6,7,8,9,10,11

مثال // رتب العناصر التالية ترتيبا تصاعديا باستخدام الترتيب السريع
(Quick Sort Algorithm)

20 ، 85 ، 60 ، 75 ، 70 ، 88 ، 50 ، 90 ، 33 ، 95

الحل// نقوم باستخدام المتغيرات التالية:

الموقع X : العنصر الموجود في وسط القائمة $X = 10/2 = 5$
القيمة : X = 70
F = Front مقدمة القائمة وتمثل بالعداد (i)
L = Last مؤخرة القائمة وتمثل بالعداد (j)

المرحلة الأولى: X = 5
20 ، 85 ، 60 ، 75 ، 70 ، 88 ، 50 ، 90 ، 33 ، 95

I=1 F=1
J=10 L=10

95 > 20 أي لا يوجد تبديل
20 ، 85 ، 60 ، 75 ، 70 ، 88 ، 50 ، 90 ، 33 ، 95
I=2 F=2
J=9 L=9
85 < 33 أي يوجد تبديل
20 ، 33 ، 60 ، 75 ، 70 ، 88 ، 50 ، 90 ، 85 ،95

I=3 F=3
J=8 L=8

أي لايوجد تبديل 60 < 90

20 ، 33 ، 60 ، 75 ، 70 ، 88 ، 50 ، 90 ، 85 ،95

I=5 F=5
J=6 L=6

أي لا يوجد تبديل 70 < 88

20 , 33 , 60 , 50 , 70 , 88 , 75 , 90 , 85 , 95

I=4 F=4
J=7 L=7

أي يوجد تبديل 75<50

20 , 33 , 60 , 50 , 70 , 88 , 75 , 90 , 85 , 95

I=5 J=5

20 , 33 , 60 , 50 , 70 , 88 , 75 , 90 , 85 , 95

المرحلة الثانية:

20 , 33 , 60 , 50 , 70 , 88 , 75 , 90 , 85 , 95

I=1 J=4 I=6 J=10

X = 33
20 , 33 , 60 , 50
→ X ←

X = 90
88 , 75 , 90 , 85 , 95
→ X ←

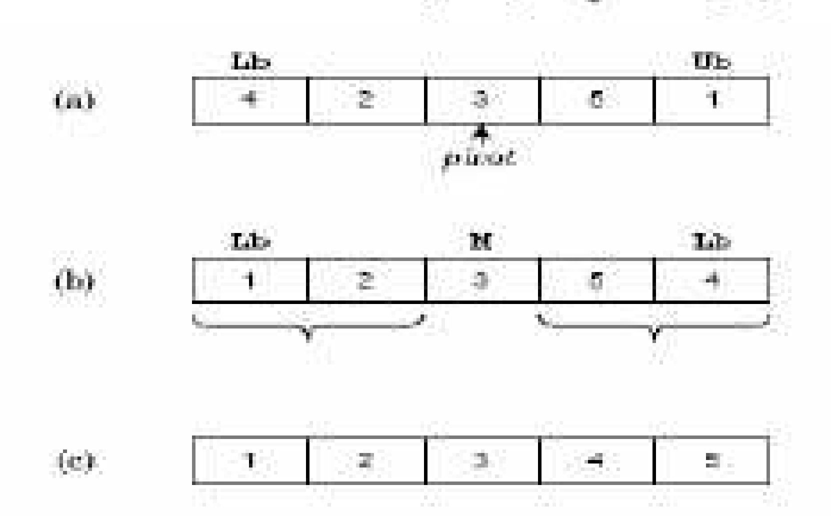
ملاحظة// أن خوارزمية الترتيب السريع تستغرق الكثير من الوقت خاصة أذا كان عدد العناصر كبير حيث أن :

$N \log_2 N$ تمثل معدل عدد المقارنات

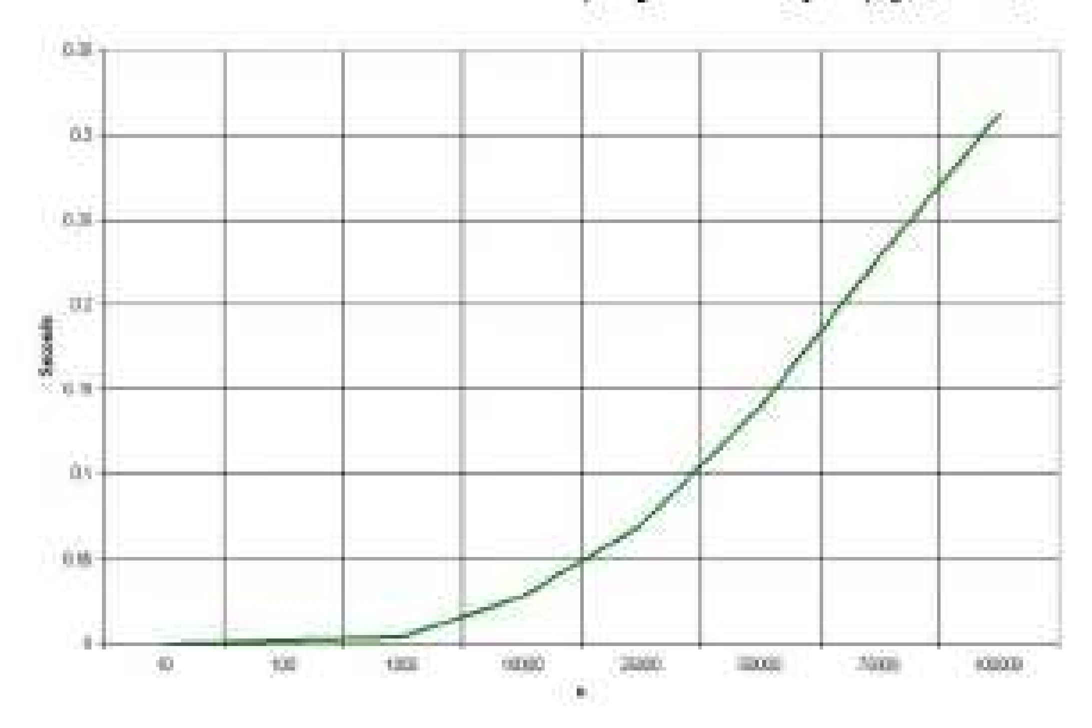
$^{1/2} N \log_2 N$ تمثل معدل التبديلات

مثال // رتب عناصر القائمة الآتية (4,3,3,5,1) باستخدام خوارزمية الترتيب السريع ؟

الحل// يمكن ذلك كما في الخطوات الآتية :



التحليل التجريبي (Empirical Analysis):



شكل(11):فعالية خوارزمية السريع (Quick Sort Efficiency)

والجزء البرمجي الخاص بتطبيق خوارزمية الترتيب السريع هو :

```
void quickSort(int numbers[], int array_size)
{
  q_sort(numbers, 0, array_size – 1);
}
void q_sort(int numbers[], int left, int right)
{
  int pivot, l_hold, r_hold;

  l_hold = left;
  r_hold = right;
  pivot = numbers[left];
  while (left < right)
  {
    while ((numbers[right] >= pivot) && (left < right))
      right--;
    if (left != right)
    {
      numbers[left] = numbers[right];
      left++;
    }
    while ((numbers[left] <= pivot) && (left < right))
      left++;
    if (left != right)
    {
      numbers[right] = numbers[left];
      right--;
    }
  }
  numbers[left] = pivot;
  pivot = left;
  left = l_hold;
  right = r_hold;
  if (left < pivot)
    q_sort(numbers, left, pivot-1);
  if (right > pivot)
    q_sort(numbers, pivot+1, right);

}
```

تحسينات أخرى:

عند استعمال الترتيب السريع لترتيب مجموعة ذات عناصر كبيرة، يمكن تغيير تقنية الترتيب عند الوصول إلى مجموعة جزئية غير مرتبة عدد عناصرها صغير أي 10عناصر أو أقل،فان الترتيب بالاختيار مناسب في هذه الحالة.

مثال// استخدام دوال تقوم بتقسيم قائمة كبيرة إلى مجاميع جزئية اصغر وترتيبها باستخدام خوارزمية الترتيب السريع .

```
typedef int tab_entiers[MAX];

int rapideEtape(tab_entiers t ،int min ،int max) {
	int temp = t[max];
	while(max > min) {
		while(max > min && t[min] <= temp) min++;
		if(max > min) {
			t[max] = t[min];
			max--;
			while(max > min && t[max] >= temp) max--;
			if(max > min) {
				t[min] = t[max];
				min++;
			}
		}
	}
	t[max] = temp;
	return max;
}

void rapide(tab_entiers t ،int deb ،int fin) {
	int mil;
	if(deb < fin) {
		mil = rapideEtape(t،deb،fin);
		if(mil - deb > fin - mil) {
			rapide(t،mil+1،fin);
			rapide(t،deb،mil-1);
		}
		else {
			rapide(t،deb،mil-1);
			rapide(t،mil+1،fin);
		}
	}
}
```

مثال// برنامج يوضح كيفية استدعاء الدوال التي تقوم بعملية الترتيب السريع (Quick Sort Algorithm) .

```
Sort(A)
      Quicksort(A,1,n)


Quicksort(A, low, high)
      if (low < high)
            pivot-location = Partition(A,low,high)
            Quicksort(A,low, pivot-location - 1)
            Quicksort(A, pivot-location+1, high)


Partition(A,low,high)
      pivot = A[low]
      leftwall = low
      for i = low+1 to high
            if (A[i] < pivot) then
                  leftwall = leftwall+1
                  swap(A[i],A[leftwall])
      swap(A[low],A[leftwall])
```

الجدول التالي يوضح وقت الترتيب لخوارزميات (الإضافة ، شيل ، الترتيب السريع) .

method	*statements*	*average time*	*worst-case time*
insertion sort	9	$O(n^2)$	$O(n^2)$
shell sort	17	$O(n^{1.25})$	$O(n^{1.5})$
quicksort	21	$O(n \lg n)$	$O(n^2)$

count	*insertion*	*shell*	*quicksort*
16	39 μs	45 μs	51 μs
256	4,969 μs	1,230 μs	911 μs
4,096	1.315 sec	.033 sec	.020 sec
65,536	416.437 sec	1.254 sec	.461 sec

6- خوارزمية الترتيب الاساسي (الرقمي) (Radix sort Algorithm):

نوع من الترتيب يعتمد على المرتبة الموجود فيها الرقم وتقسم إلى خانات بحيث ترتب العناصر حسب المراتب (Pockets) ،في هذه الطريقة يستخدم ما يسمى بالخانات (Digit) (9....0) ،تعاد العملية في كل مرتبة بحيث يكون عدد المراحل مساوي إلى أكبر عدد للمراحل في أكبر رقم .

مثال تطبيق // أذا كان لدينا العناصر التالية (9 , 7 , 132) ، فان أكبر رقم يحتوي على 3 مراتب هو (132)

ملاحظة: أن هذه الطريقة تعتبر غير عملية وذلك لإعادة استخدامها والاختبار في كل مره بحيث تعتبر (link Queue) ، أي أن كل خانة هي بمثابة طابور متصل FIFO

مثال// لديك العناصر التالية رتبها باستخدام الترتيب الرقمي (Radix sort Algorithm):

42	23	74	11	65	58	94	36	99	87

Ordinal

الحل// يمكن توضيحه بجدول لكل خانة (مرتبة) وكما يلي:

0	1	2	3	4	5	6	7	8	9
-	11	42	23	74,94	65	36	87	58	99

1- Pockets1 11,42,23,74,94,65,36,87,58,99

0	1	2	3	4	5	6	7	8	9
	11	23	36	42	58	65	74	87	94,99

2- Pockets2 11,23,36,42,58,65,74,87,94,99

7- ترتيب المؤشرات (Sorting Pointers Algorithm):

تستخدم هذه الطريقة لربط وترتيب العناصر حسب المؤشرات حيث تستخدم فكرة التبديل كما في المثال التالي .

مثال// رتب العناصر الآتية بطريقة ترتيب المؤشرات (Sorting Pointers Algorithm)

15 , 37 ,9 , 60 , 43

الحل// يمكن توضيحه بالخطوات الآتية :

أ- 1 – ضع العناصر في خانات
2 – ضع المؤشرات

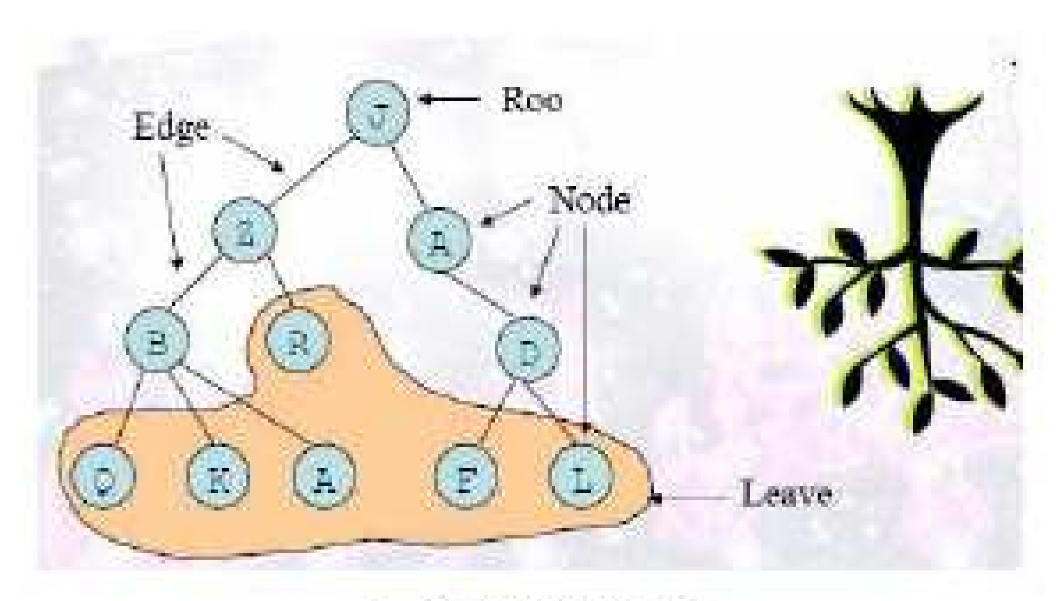
15	←	1
37	←	2
9	←	3
60	←	4
43	←	5

ب - ترتيب المؤشرات

15		1
37		2
9		3
60		4
43		5

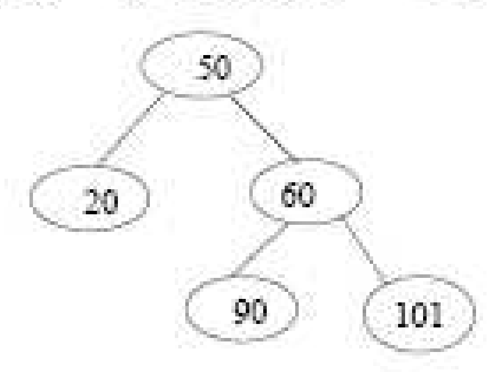
8- الترتيب الشجري لشجرة البحث الثنائية (Tree for Binary search tree):

الشجرة هي التي تكون فيها القيمة البيانية لأي عقدة أكبر من الفرع الأيسر لها وأصغر من قيمة الفرع الأيمن لها ،وتتألف من الجذر (Root) والعقد (Nodes) والأوراق (Leaves). وتبنى وفق خطوات معينة وكذلك تحذف بخطوات معينة أيضا ، والشكل رقم (12) الآتي يوضح الهيكل الشجري.



شكل رقم (12) الترتيب الشجري

مثال تطبيق// شجرة بحث ثنائية فيها قيم بيانية مرتبة حسب أولويات البناء الشجري .



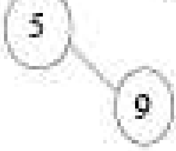
مثال// كون شجرة بحث ثنائية (Tree Binary Search) للعناصر التالية مفسرا كل خطوة ؟
5,9,7,3,8,12,6,4,20

الحل// يمكن ذلك من خلال الخطوات الآتية:

1 – نأخذ العنصر الأول فيكون جذر

5

2 – نأخذ العنصر الثاني فيكون فرع أيمن لأنه أكبر من الجذر



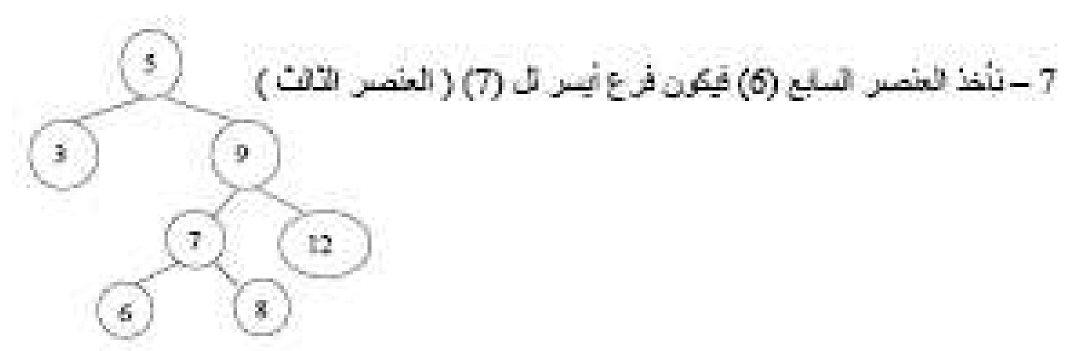
3 – نأخذ العنصر الثالث فيكون فرع أيسر للعنصر الثاني

4 – نأخذ العنصر الرابع فيكون فرع أيمن للجذر

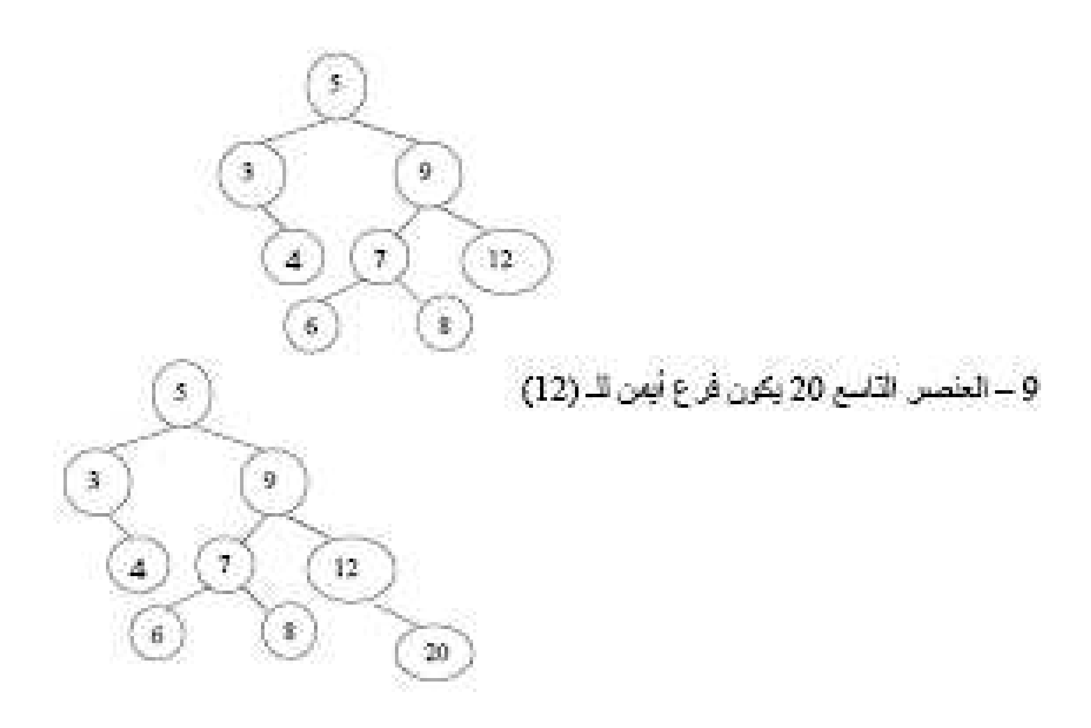
5 – نأخذ العنصر الخامس فيكون فرع أيمن (7) (العنصر الثالث)

6 – نأخذ العنصر السادس فيكون فرع أيمن لل(9) (العنصر الثاني)

7 – نأخذ العنصر السابع (6) فيكون فرع أيسر لل (7) (العنصر الثالث)



8 – العنصر الثامن (4) يكون فرع أيمن لل(3) (العنصر الرابع)



9 – العنصر التاسع 20 يكون فرع أيمن للـ (12)

3 , 4 , 5 , 6 , 7 , 8 , 9 , 12 , 20

أما بالنسبة لعملية الحذف الخاصة بالأشجار الثنائية فانه يمكن توضيحها بالطرق الثلاث الآتية:

1- حذف عقدة نهائية (ورقة) (Leave Delete) :
لكي نقوم بحذف عقدة نهائية فإننا يجب أن نلغي العقدة بدون أن نؤشر على بقية العقد

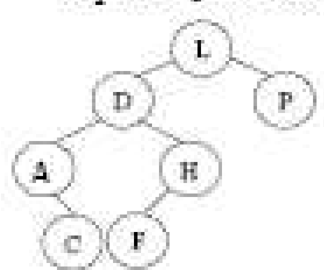


قبل الحذف بعد الحذف

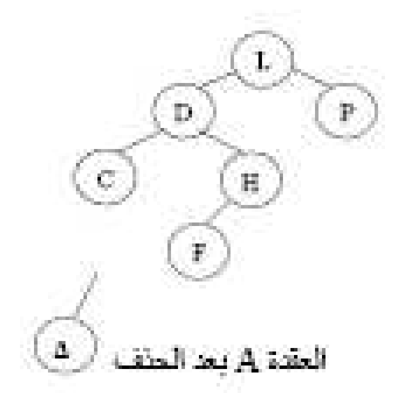
2- حذف عقدة لها ابن واحد (One Child Node Delete):

لكي نقوم بعملية الحذف هذه نطبق ما يلي:
أ - نجعل المؤشر أي العقدة يشير إلى العقدة الابن

ب - تلغي العقدة المقصودة dispose



العقدة A قبل الحذف



العقدة A بعد الحذف

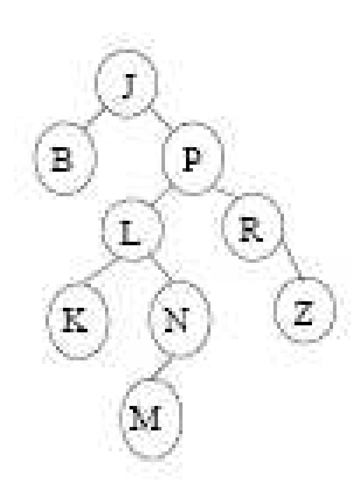
3- حذف عقدة لها فرعان (Two Childs Node Delete):

يتم ذلك من خلال الخطوات التالية:

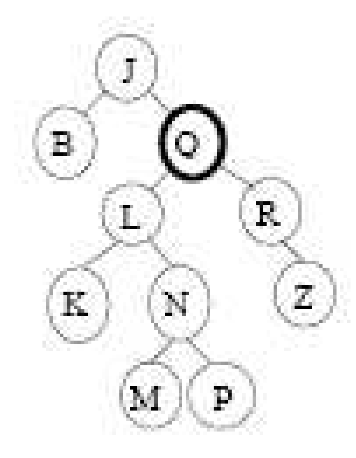
أ - تستبدل العقدة المطلوب حذفها بالعقدة التالية لها بالقيمة وهذه ستحصل عليها من الشجرة الفرعية اليسرى أو الشجرة الفرعية اليمنى بالنسبة للعقدة .

ب - تأخذ الشجرة الفرعية اليسرى للعقدة أي العقدة في سياق العقدة المطلوب حذفها ، علما أنه أذا لم يكن لها فرع أيسر فيكون الفرع الأيمن بديل .

الشجرة التالية توضح عملية حذف العقدة Q واستبدالها بالعقدة P .



العقدة Q بعد الحذف



العقدة Q قبل الحذف

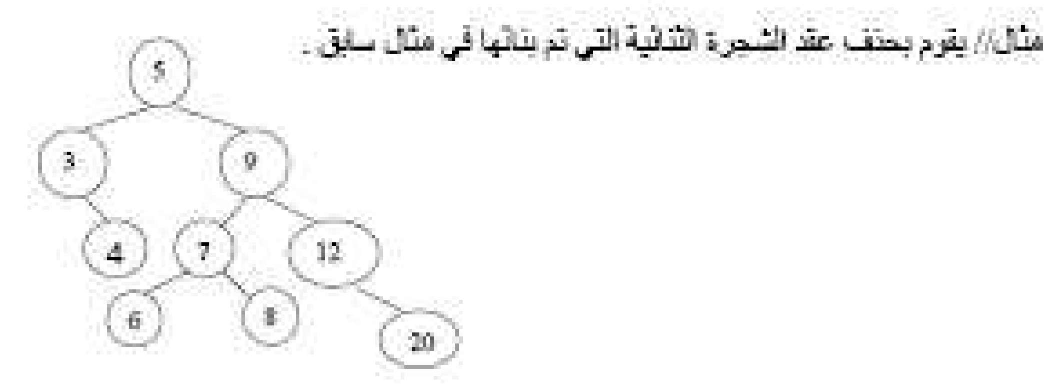
الخوارزمية الخاصة بحذف عقدة في شجرة البحث الثنائية :

```
Tree-Delete(T, z)
        if (left[z] = NIL) or (right[z] = NIL)
                then y ← z
                else y ← Tree-Successor(z)
        if left[y] ≠ NIL
                then x ← left[y]
                else x ← right[y]
        if x ≠ NIL
                then p[x] ← p[y]
        if p[y] = NIL
                then root[T] ← x
                else if (y = left[p[y]])
                        then left[p[y]] ← x
                else right[p[y]] ← x
    if (y <> z)
        then key[z] ← key[y]
                /* If y has other fields, copy them, too. */
    return y
```

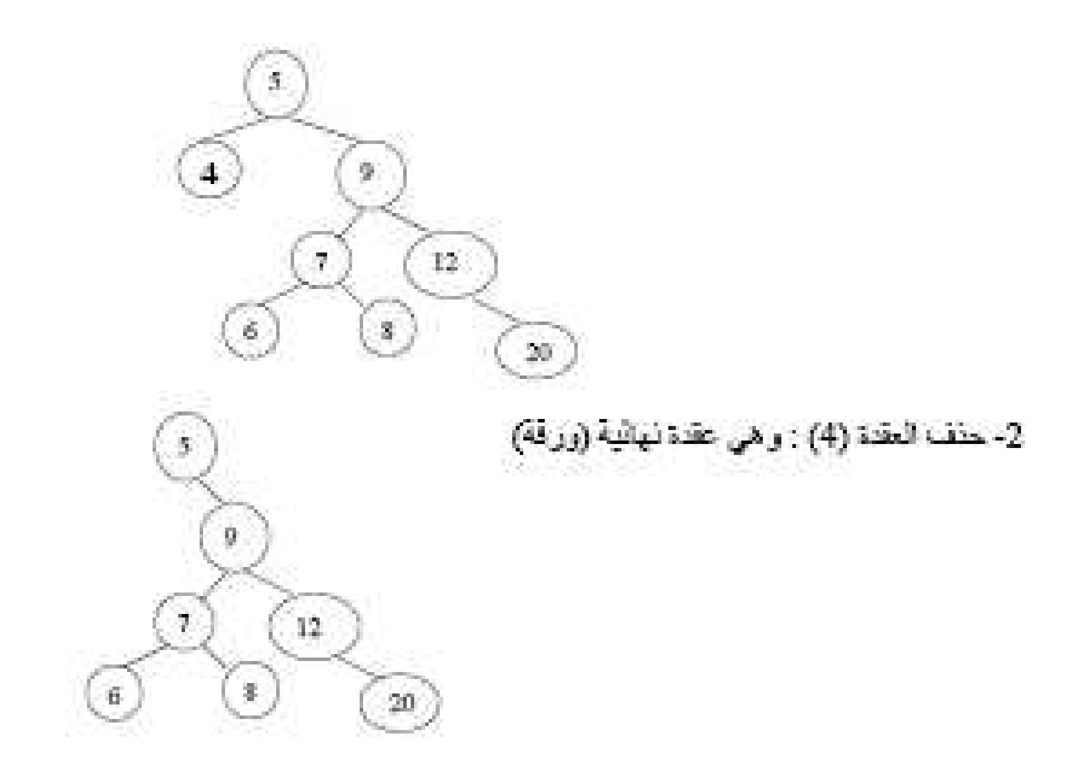
* في حالة أنه الفرع الأيسر للعقدة لا يحتوي على عقدة أقصى اليمين فأن العقدة اليسرى هي اليمين .

* أذا كانت العقدة التي لها شجرة فرعيه يمنى تعبر هذه الحالة أي تملك ابن واحد في أقصى الشجرة فان العقدة المراد حذف 3 هي عقدة لها أبن واحد لذا فأن الابن يكون بدلا عنها

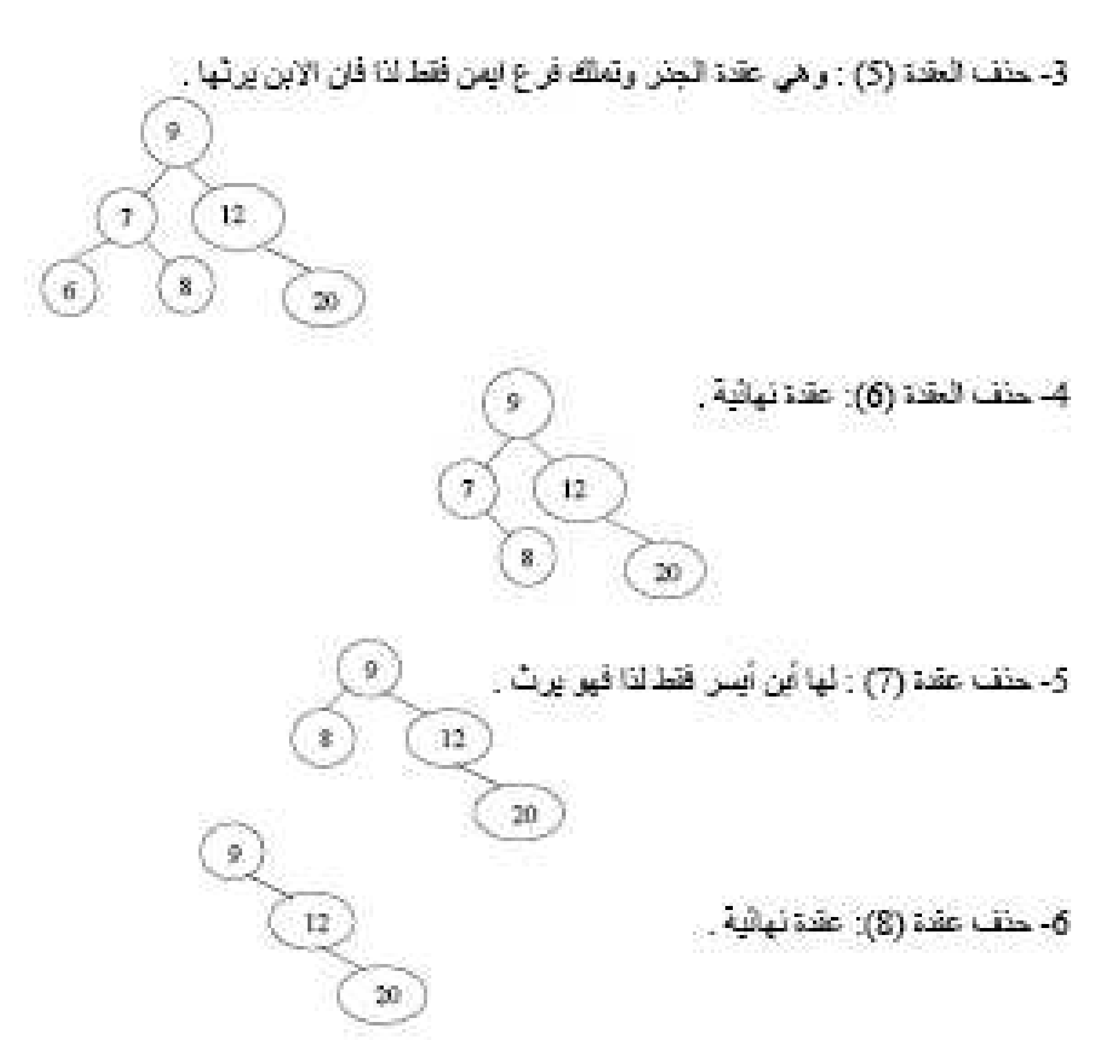
مثال// يقوم بحذف عقد الشجرة الثنائية التي تم بنائها في مثال سابق .



1- حذف العقدة (3) : تملك ابن أيمن واحد لذا فهو يرثها



2- حذف العقدة (4) : وهي عقدة نهائية (ورقة)



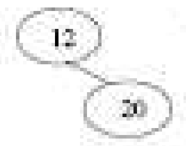
3- حذف العقدة (5) : وهي عقدة الجذر وتملك فرع ايمن فقط لذا فان الابن يرثها .

4- حذف العقدة (6): عقدة نهائية .

5- حذف عقدة (7) : لها ابن أيسر فقط لذا فهو يرث .

6- حذف عقدة (8): عقدة نهائية .

7- حذف عقدة (9): تملك فرع أيمن لذا فهو يرثها .

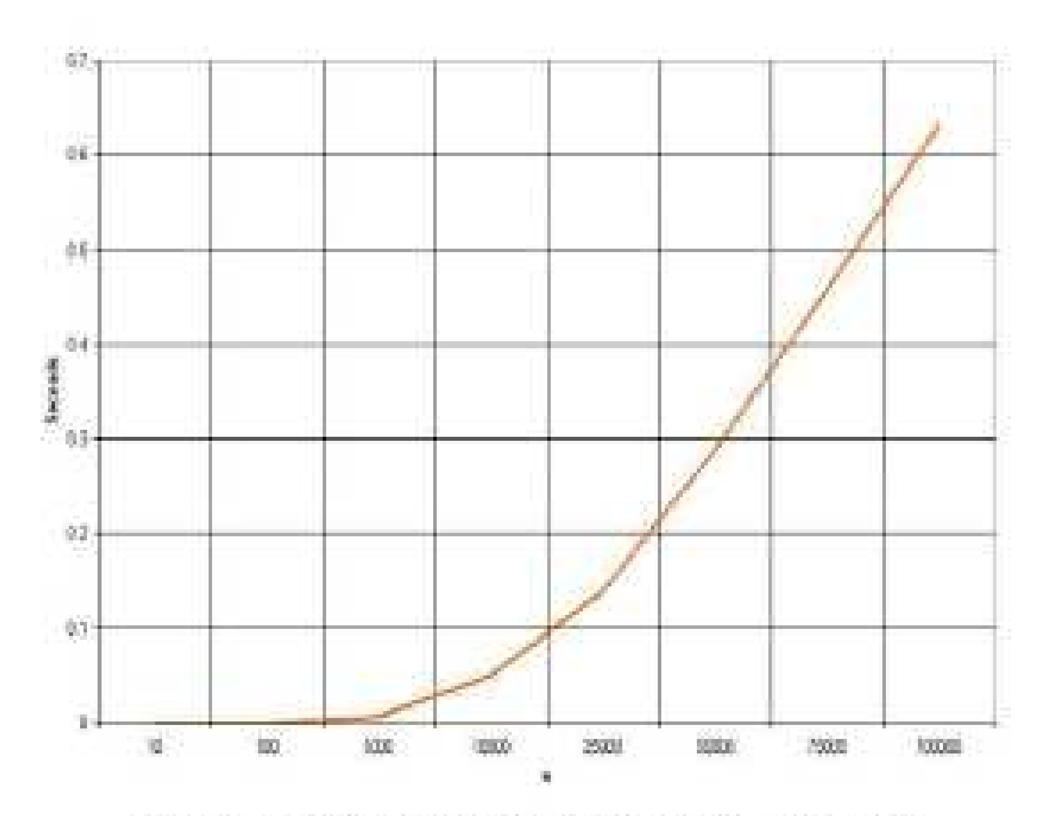


8- حذف عقدة (12) : تملك فرع ايمن يرثها .

وتوضح هذه العقد بخزنها داخل المكدس كالآتي :

3	4	5	6	7	8	9	10	11

التحليل التجريبي (Empirical Analysis):



شكل(13):فعالية خوارزمية الترتيب الشجري(Tree Sort Efficiency)

وجزء البرنامج الذي يقوم بعملية الترتيب الشجري هو:

```
void TreeSort(int numbers[], int array_size)
{
  int i, temp;
  for (i = (array_size / 2)-1; i >= 0; i--)
   siftDown(numbers, i, array_size);
  for (i = array_size-1; i >= 1; i--)
  {
    temp = numbers[0];
    numbers[0] = numbers[i];
    numbers[i] = temp;
    siftDown(numbers, 0, i-1);
  }
}

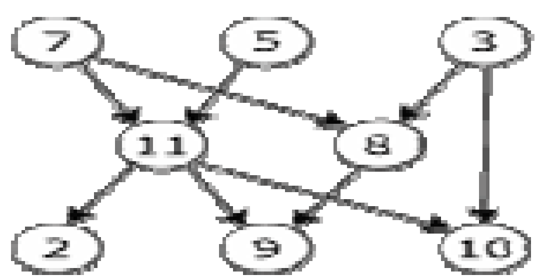

void siftDown(int numbers[], int root, int bottom)
{
  int done, maxChild, temp;
  done = 0;
  while ((root*2 <= bottom) && (!done))
  {
    if (root*2 == bottom)
      maxChild = root * 2;
    else if (numbers[root * 2] > numbers[root * 2 + 1])
              maxChild = root * 2;
          else
              maxChild = root * 2 + 1;

    if (numbers[root] < numbers[maxChild])
    {
      temp = numbers[root];
      numbers[root] = numbers[maxChild];
      numbers[maxChild] = temp;
      root = maxChild;
    }
    else
      done = 1;
  }

}
```

9- خوارزمية الترتيب التبولوجي (Topological sorting Algorithm):

تستعمل خوارزمية الترتيب هذه فكرة المخططات ووضع القيم بالطابور،ويمكن توضيحها بالمثال التالي :



The graph shown to the left has many valid topological sorts, including:

- 7,5,3,11,8,2,10,9
- 7,5,11,2,3,10,8,9
- 3,7,8,5,11,10,9,2
- 3,5,7,11,10,2,8,9

من محاسن الطريقة هي أن وقت التنفيذ خطي يتمثل بزيادة العقد بين المسارات الحافات ، وقد استخدمت إحدى الخوارزميات كانت من قبل العالم (Kahn 1962) معتمدة فكرة الطابور

حيث إن :

E= الحافة

Q=الطابور

V=الرؤوس

والخوارزمية التي توضح الطريقة هي :

```
L ← Empty list where we put the sorted elements
Q ← Set of all nodes with no incoming edges
while Q is non-empty do
   remove a node n from Q
   insert n into L
   for each node m with an edge e from n to m do
      remove edge e from the graph
      if m has no other incoming edges then
         insert m into Q
if graph has edges then
   output error message (graph has a cycle)
```

```
else
   output message (proposed topologically sorted order: L)
```

4-2 خوارزميات الترتيب الخارجي (External Sorting Algorithms):

1- خوارزمية ترتيب الدمج (Merge sort Algorithm):

مثال// يقوم بترتيب مجموعة عناصر بطريقة الدمج ،كما في الخطوات الآتية:

phase	T1	T2	T3
A	7 6 8 4	9 5	
B	7 6		9 8 5 4
C	9 8 5 4	7 6	
D			9 8 7 6 5 4

وجزء البرنامج الخاص بتطبيق طريقة ترتيب الدمج هو:

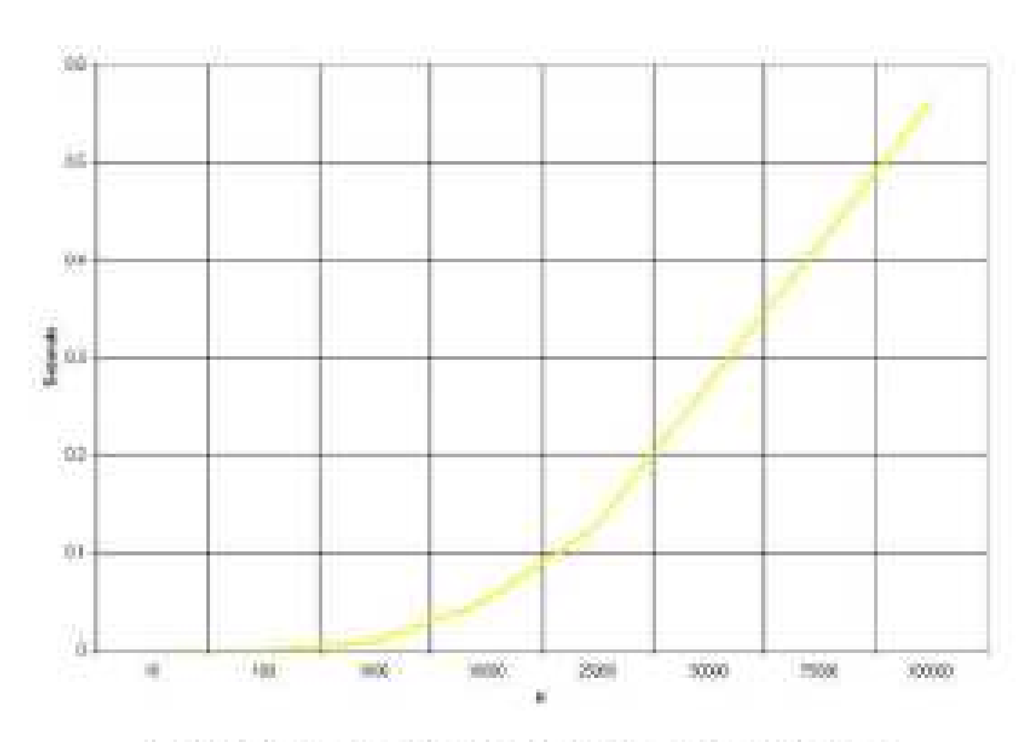
```
void MrgeSort(int numbers[], int temp[], int array_size)
   {
 m_sort(numbers, temp, 0, array_size - 1);
}
void m_sort(int numbers[], int temp[], int left, int right)
{
 int mid;
 if (right > left)
  {
   mid = (right + left) / 2;
   m_sort(numbers, temp, left, mid);
   m_sort(numbers, temp, mid+1, right);
   merge(numbers, temp, left, mid+1, right);
  }
```

```
}
void merge(int numbers[], int temp[], int left, int mid, int right)
{
  int i, left_end, num_elements, tmp_pos;
  left_end = mid - 1;
  tmp_pos = left;
  num_elements = right - left + 1;

  while ((left <= left_end) && (mid <= right))
  {
    if (numbers[left] <= numbers[mid])
    {
      temp[tmp_pos] = numbers[left];
      tmp_pos = tmp_pos + 1;
      left = left +1;
    }
    else
    {
      temp[tmp_pos] = numbers[mid];
      tmp_pos = tmp_pos + 1;
      mid = mid + 1;
    }
  }
  while (left <= left_end)
  {
    temp[tmp_pos] = numbers[left];
    left = left + 1;
    tmp_pos = tmp_pos + 1;
  }
  while (mid <= right)
  {
    temp[tmp_pos] = numbers[mid];
    mid = mid + 1;
    tmp_pos = tmp_pos + 1;
  }
  for (i=0; i <= num_elements; i++)
  {
    numbers[right] = temp[right];
    right = right - 1;
  }

}
```

التحليل التجريبي (Empirical Analysis) :



شكل(14):فعالية خوارزمية الدمج (Merge Sort Efficiency)

2- خوارزمية ترتيب الدمج المتوازن ذي المسارين
:(Balanced Two - way - Merge Sort)

يمكن توضيح فكرة هذه الطريقة بالمثال الآتي الخاص بقائمة معينة كما في الخطوات التالية:

1- تقسم القائمة الأصلية إلى قائمتين متساويتين تقريبا ولتكن A,B ونضع كل عنصر من B مع نظيرة الأول في القائمة A

2- نقارن العنصر الأول في القائمة B مع العنصر نظيرة الثاني في القائمة A و نضعه في القائمة C بالترتيب.

3- نقارن العنصر الثاني في القائمة B مع العنصر نظيرة الثاني في القائمة A ونضعة بالقائمة D في الترتيب .

4- نكرر الخطوتين 2، 3 لنحصل على عناصر طولها 2 في كل من القائمتين D,C ونضع العناصر بالترتيب في القائمتين B,A

5- بنفس الطريقة نقوم بدمج عناصر القائمتين A,B حيث عناصرها بطول 4 لتكون مرتبة ونضعها في القائمتين C,D

6- نعيد الطريقة بدمج عناصر القائمتين B,A بطول 8 .

7- نستمر بهذا الأسلوب لحين الوصول إلى قائمة مرتبة

مثال // يقوم بترتيب العناصر الآتية باستخدام طريقة ترتيب الدمج ذو المسارين
.(Balanced Two - way - Merge Sort)

(18 ، 23 ، 2 ، 50 ، 42 ، 63 ، 20 ، 28 ، 33 ، 47 ، 3)

الحل// يمكن توضيح ذلك بالخطوات الآتية:

1- N = 11   A = 18 , 23 , 2 , 50 , 42 ,
B = 63 , 20 , 28 , 33 , 47 , 3

2- فردية   C = 18 , 63 ,   , 2 , 28 ,   , 42 , 47
زوجية   D = 20 , 23 ,   , 33 , 50 ,   , 3

تكون قائمتين للتسلسلات الزوجية D والتسلسلات الفردية C

3-   A = 18 , 20 , 23 , 63 ,   , 3 , 42 , 47
B = 2 , 28 , 33 , 50 ,

4-   C = 2 , 18 , 20 , 23 , 28 , 33 , 50 , 63
D = 3 , 42 , 47

5-   A = 2 , 3 , 18 , 20 , 23 , 28 , 33 , 42 , 47 , 50 , 63   مرتبه

3- خوارزمية ترتيب الدمج باستخدام طريقة قسم وأنتصر
: (Divided and Conquer Merge Sort Algorithm)

باستخدام طريقة قسم وأنتصر أو فرق تسد ، حيث تقسم القائمة المعطاة إلى مجاميع فرعية من القوائم، كل قائمة تكون من عنصرين بعد ذلك نقوم بدمج القوائم الفرعية المستخدمة ويمكن توضيحها كما في المثال الآتي:

مثال// استخدام طريقة فرق تسد في ترتيب الدمج للقائمة التالية:

(43 ، 54 ، 11 ، 41 ، 93 ، 17 ، 5 ، 60)

الحل// يمكن توضيح ذلك كما في الخطوات الآتية:

1- نجد عدد عناصر القائمة الرئيسية N = 8
2- نقسمها إلى قوائم فرعية كل أثنين في قائمة
3- نقوم بدمج كل قائمتين سوية وترتبها
4- نعيد الخطوة السابقة للقوائم المتبقية وترتبها

[43 , 54] , [11 , 41] , [93 , 17] , [5 , 66]

2   1

[11 , 41 , 43 , 54] , [5 , 17 , 66 , 93]

5 , 11 , 17 , 41 , 43 , 54 60 , 93   قائمة مرتبة

هذه الطريقة لها مساوئ هي :
1- إنها تحتاج إلى مصفوفة خزن أضافية يمكن أن تكون أكبر من المصفوفة الأصلية (أذا كان عدد العناصر فردي).
2- تحتاج إلى مصفوفات فرعية عددها كبير ، هذه المصفوفات تفصل أولا ، ثم تدمج وهذا يعني أن الوقت المحتاج لإكمال عملية الترتيب كبير نسبة إلى غيرها من الطرق.
إذ نستنتج من ذلك بأن:

عدد المراحل   No. Of Pass = log n
عدد المقارنات   No. Of Compaction = n log n

الفصل الثالث

البحث

Searching

3-1: البحث (Search):

البحث هي عملية إيجاد عنصر معين في مجموعة من البيانات فإذا كان العنصر موجود في المجموعة العملية تعتبر ايجابية وإلا ستكون سلبية في حالة عدم وجوده ، ولكي تكون العملية فعالة يفضل إن تكون العناصر مرتبة.

3-2: البحث التسلسلي (Sequential Search):

هي عملية البحث عن عنصر من خلال مسح أو استعراض عناصر القائمة من بدايتها وبالتسلسل لحين الوصول للعنصر المطلوب إذا كان موجودا أما في حالة الوصول لنهاية القائمة ولم نحصل علية يعني إن العنصر غير موجود.

عدد المقارنات=n/2

وقت التنفيذ=O(n)

مثال// البرنامج التالي يقوم بالبحث عن عنصر من بين مجموعة عناصر باستخدام البحث التسلسلي علما أن عدد العناصر (7) والعنصر المراد البحث عنه (3) والعناصر هي :

data[ ]={7,4,5,6,3,9,10}

الحل//

```
#include<stdio.h>
main( )
{
Int data[ ]={7,4,5,6,3,9,10};
Int nmax=7;
Int key=3;
Printf("%d\n",sqsearch(data,nmax,key));
}
Sqsearch(data,n,k);
Int data[ ];
Int n;
Int k;
{
  Register int i;
   For (i=0;i<=n;++i)
  If (k==data[ I ]))return(i+1);
 Return(-1);   / * no match exist * /
}
```

نتيجة البحث هي إن العنصر (3) موجود في القائمة بالموقع (5).

3-3: البحث الثنائي (Binary search):

تقوم فكرة البحث الثنائي على تقسيم المصفوفة إلى نصفين واستبعاد النصف الذي لا ينتمي إليه المفتاح key الذي نبحث عنه عن طريق تحديد العنصر الذي يقع في منتصف هذه المصفوفة، ثم نقارن هذا العنصر مع المفتاح الذي نبحث عنه (المصفوفة مرتبة أبجدياً).
والـ Pseudo code التالي يوضح لنا هذه الطريقة:

```
repeat:
If  ID <= Array[k] then j=k-1
If  ID >= Array[k] then i=k+1
until i>=j
If  i-1>j then we found the ID in the array
else the ID is not found
```

مثال // مصفوفة مرتبة ترتيباً أبجدياً:

```
word[]={"begin", "const", "do", "end", "if", "odd", "program", "read",
         "then", "var", "while", "write"}
```

كيف نكتب code لخوارزمية البحث الثنائية ؟ كيف نبحث في المصفوفة أبجدياً؟
يتم مقارنة عنصران أبجدياً عن طريق دالة خاصة بمقارنة السلاسل الحرفية هي:

```
strcmp (char *str1, char *str2)
```

هذه الدالة تقوم بمقارنة حرفين أو سلسلتين حرفيتين، وتقوم بإرجاع:

القيمة (صفر) إذا كانت	str1= str2
قيمة سالبة إذا كانت	str1< str2
قيمة موجبة إذا كانت	str1> str2

وجزء البرنامج الذي يقوم بتطبيق خوارزمية البحث الثنائي (binary search) هو :

```
#include "STRING.H"
#include "STDIO.H"
#define max_size_12
//-----------------------------------
int binary_search (ID);
char *ID;
{
char *word[]={"begin", "const", "do", "end", "if", "odd", "program",
               "read", "then", "var", "while", "write"};
int i=0, j=max_size-1, s, k;
while(i<=j)
  {
   k=(i+j)/2;
   s=strcmp(ID, word[k]);
   if (s<=0) j=k-1;
   if (s>=0) i=k+1;
  }
```

```
  if((i-1)>j) {
    printf("nwe found the key (%s) at element %i", word[k], k+1);
    return k;
  }
  return -1;
}
//--------------------------------
void main()
{
int result;
char *ID;
printf("nnPlz. Enter the ID to begin searchnnID=");
scanf("%s",ID);
result = binary_search(ID);
if (result==-1) {
printf("nthe key(%s) is not found",ID); }
getch();
}
```

ولكي نطبق خوارزمية البحث الثنائي (Binary Search)على مصفوفة ما نتبع الخطوات البسيطة التالية:

1- الخطوة الأولى والأهم والتي لا يمكن تطبيق الخوارزمية إلا إذا كانت العناصر مرتبة تصاعدياً أو تنازلياً أو أبجدياً على حسب نوع البيانات المخزنة فيها .

2- تحديد أول عنصر في المصفوفة المعرف i ، وآخر عنصر فيها والمعرف مثلاً j .

3- تحديد العنصر الذي يقع في منتصف هذه المصفوفة المعرف k.

4- بعد ذلك يمكننا تطبيق البحث الثنائي على مصفوفتنا فإذا كان :

أ- إذا كان يساويه نكون قد وجدنا العنصر الذي نبحث عنه.

ب- إذا كانت قيمة المفتاح أقل من قيمة العنصر الأوسط في المصفوفة، إذن نحتاج أن نبحث فقط في نصف المصفوفة الأول ونستبعد البحث في نصفها الثاني.

وفيما عدا ذلك إذا كانت قيمة المفتاح أكبر من قيمة العنصر الأوسط في المصفوفة، إذن نحتاج أن نبحث فقط في نصف المصفوفة الثاني ونستبعد البحث في نصفها الأول.

حيث نعتبر النصف الذي حددنا البحث فيه مصفوفة قائمة بحد ذاتها، نحدد فيها الـ i, j, & k (أي نقوم بتقسيمها إلى قسمين) ونطبق نفس الخطوات من 1 إلى 3 فيها، ثم نقارن المفتاح مع العنصر الأوسط الجديد، بنفس الترتيب الذي ذكر في الخطوات 1 إلى3 السابقة.

أن خوارزمية هذا البحث تقوم بالبحث عن عنصر في قائمة مرتبة

1- تحديد موقع العنصر الذي يقع في منتصف القائمة تقريبا

2- أذا كان العنصر المطلوب(Item) مساويا عنصر في الوسط (X) معنى ذلك أنه عملية البحث مع العنصر الذي في الوسط انتهت، إما إذا كانت أقل من قيمة العنصر في الوسط فسينحصر البحث في الجهة اليسرى والأقل من قيمة العنصر (X) ،أما أذا كان العنصر الذي نبحث عنه اكبر من قيمة (X) فيكون البحث في الجهة اليمنى والأكبر.

3- في الحالتين أعلاه فانه تتم المعالجة بنفس الطريقة التي تمت فيها المقارنة السابقة لحين الوصول إلى العنصر المطلوب أي أن

عدد العناصر هو: N

عدد المقارنات هو : $\log_2 n$

Item = 30   أصغر ← X → أكبر

1- Item = X	انتهت العملية
2- Item < X	البحث في القسم الأيسر
3- Item > X	البحث في القسم الأيمن

مثال// نفترض أننا نبحث عن عناصر مختلفة في هذه المصفوفة :

Array[]={0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28}

و يجب ترتيب العناصر قبل البدء بالبحث.

الحل// يمكن توضيحه بالبرنامج الآتي :

```
#include "STRING.h"
#include "STDIO.h"
#define max_size 15
//--------------------------------
int binary_search (key);
int key;
{
int NumArray[]={0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28};
int i=0, j=max_size-1, k=(i+j)/2;
while(i<J){
    if (key = NumArray[k]){
    printf("nwe found the key (%i) ", key);
    return k;
        }
   else{
        if (key < NumArray[k]){
          j=k;
          k=(i+j)/2;
              }
       if (key > NumArray[k]){
         i=k;
         k=(i+j)/2;
            }
       }
  }
  return -1;
```

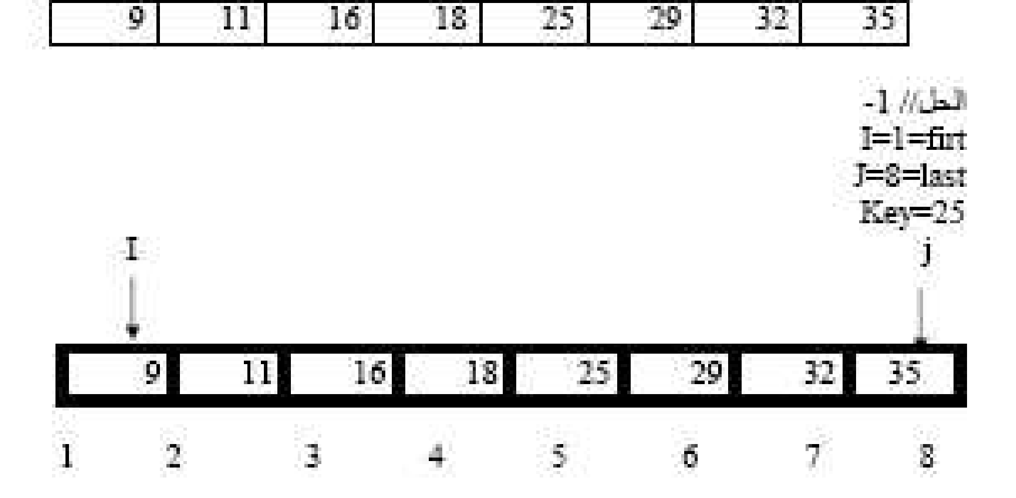
```
}
//--------------------------------
void main()
{
int result;
int key;
printf("nnPlz. Enter the Key to begin searchnKey=");
scanf("%i",key);
result = binary_search(key);
if (result==-1){
printf("nthe key(%i) is not found",key); }
getch();
}
```

عدد مرات البحث في أي مصفوفة عن عنصر محدد باستخدام البحث الثنائي (Binary Search):

أن أقصى عدد من مرات البحث باستخدام الـ Binary Search في أي مصفوفة يُعطى من إيجاد القوة التي يرفع إليها رقم (2) بكي يعطينا العدد الذي يزيد عن عناصر المصفوفة بواحد، أي أنه أول قوة لـ (2) والتي تُعطي رقم أكبر من عدد عناصر المصفوفة بواحد.

ففي مثالنا: استخدمنا مصفوفة من (15) عنصر، نلاحظ أن العدد الذي يزيد على عدد عناصر المصفوفة بواحد، أي العدد (16) ينتج من القوة الرابعة لرقم (2) أي (2^4=16) وذلك يعني أننا نحتاج على الأكثر لأربع مرات مقارنة في الـ Binary Search حتى نجد العنصر الذي نبحث عنه، فمن الممكن أن نجده من أول مرة في المقارنة أو نجده في ثاني مرة، أو ثالث مرة أو رابع مرة.. أو إن يكون غير موجود في المصفوفة

عدد المقارنات هي : log2 n
مثال// لديك القائمة التالية المطلوب البحث عن العنصر key=25 باستخدام طريقة البحث الثنائي (Binary Search)؟



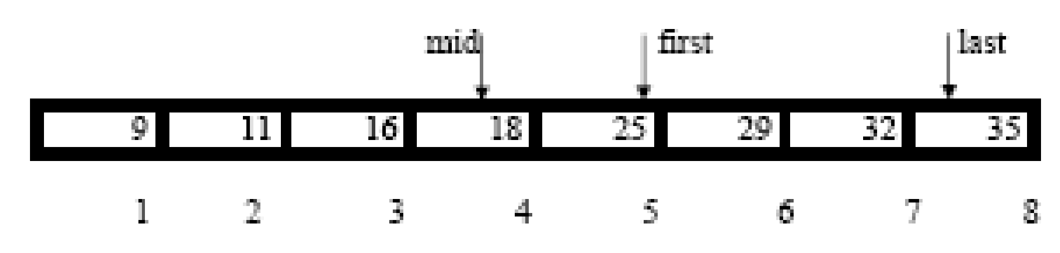
9	11	16	18	25	29	32	35

الحل// 1-
I=1=firt
J=8=last
Key=25

-2 نجد القيمة الوسطية: mid=(1+8) div 2=4

List[4]< 25

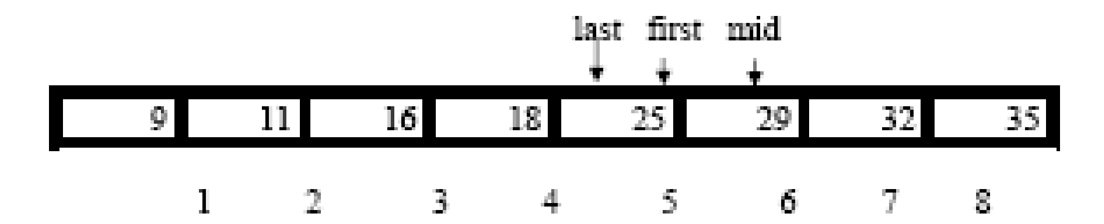
-3 اذن يكون first = mid+1



-4 نجد mid=(5+8) div 2= 6

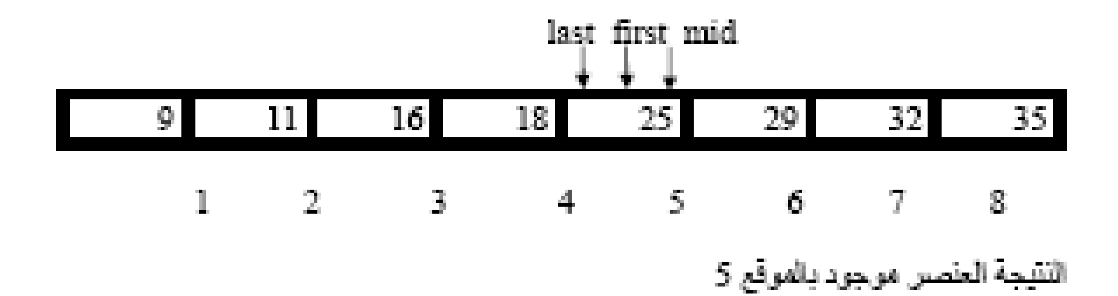
List[6]>25

Last=mid-1



-5 نجد mid=(5+5) div 2= 5

List[5]=25



النتيجة العنصر موجود بالموقع 5

مثال// استخدم الجداول للبحث عن العناصر التالية بطريقة البحث الثنائي

Data[ ]=-15,-6,0,7,9,23,54,82,101

الحل // العناصر المراد البحث عنها هي:

X=101 ,x=-14 ,x=82

X=101	Low=i=first	High=j=last	mid
	1	9	5
	6	9	7
	8	9	8
	9	9	9

Mid=(low+high) div 2

Found=101 في الموقع 9

X=-14	Low=i=first	High=j=last	mid
	1	9	5
	1	4	2
	1	1	1
	2	1	Not found

شرط التوقف هنا إن low> high

X=82	Low=i=first	High=j=last	mid
	1	9	5
	6	9	7
	8	9	8

العنصر x=82 موجود في الموقع 8

يمكن إيجاد معدل عدد المقارنات الكلية للعناصر باستخدام القانون التالي:

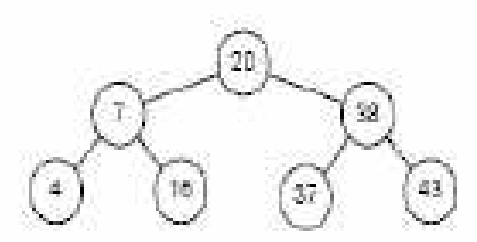
Average of comparison=sum of comparisons/number of elements

	1	2	3	4	5	6	7	8	9
element	-15	-6	0	7	9	23	54	82	101
comparisons	3	2	3	4	1	3	2	3	4

Average of comparison=25/9=2.77

4-4: البحث في الشجرة الثنائية (Binary Tree Search):

شجرة البحث الثنائية هي الشجرة التي كل عقدة فيها اكبر من ابنها في اليسار منها واقل من ابنها الذي في اليمين منها، الشكل أدناه يوضح ذلك:



شكل(15):شجرة بحث ثنائية

بعد استخدام الشجرة الثنائية لترتيب قائمة من الأعداد، يمكن البحث عن عدد ما وحذفه بحذف العقدة التي تحويه وذلك وفق ما يلي:

-إذا كانت العقدة المراد حذفها ورقة، يمكن حذفها دون تغيير آخر في الشجرة.

-إذا لم تكن العقدة المراد حذفها ورقة، يجب بعد حذفها وضع مكانها العقدة التي تحوي العدد التالي وفقاً للترتيب التصاعدي للأعداد الموجودة في الشجرة.

وجزء البرنامج الخاص بتطبيق البحث في الشجرة الثنائية هو :

```
#include<iostream.h>
struct nodetype
 {
 int k;
 struct nodetype* left;
 struct nodetype* right;
 };
typedef struct nodetype* nodeptr;
nodeptr maketree(int x)
 {
 nodeptr p;
 p=new nodetype;
 p->k=x;
 p->left=NULL;
 p->right=NULL;
 return (p);
 }
```

```
void build(nodeptr node,int number)
{
 if(number>node->k)
  if(node->right==NULL)
   node->right=maketree(number);
  else
   build(node->right,number);
 else
 {
  if(number<node->k)
   if(node->left==NULL)
        node->left=maketree(number);
   else
        build(node->left,number);
  else
   cout<<"\nDuplicate number "<<number;
 }
 return;
}
void search(int key,nodeptr root)
{
 nodeptr p,f,q,rp,s;
 p=root;  // p will point to the node
 q=NULL;  // and q to its father, if any.
 while(p!=NULL && p->k!=key)
 {
  q=p;
  if(key<p->k)
   p=p->left;
  else
   p=p->right;
 }
 if(p==NULL)
  cout<<"The key does not exist in the tree\n";  //leave the tree
unchanged
 else// rp will point to the node that will replace node p
  if(p->left==NULL) // node p has right son only
   rp=p->right;
  else
   if(p->right==NULL)// node p has left son only
        rp=p->left;
   else // node p has two sons
```

```
{
        f=p;
        rp=p->right;
        s=rp->left;
        while(s!=NULL)
        {
         f=rp;
         rp=s;
         s=rp->left;// s is always the left son of rp
        }  // now, rp is the inorder successor of p
        if(f!=p)// if p is not the father of rp
        {
         f->left=rp->right;
         rp->right=p->right;
        }
        rp->left=p->left;
  }
 if(q==NULL) // if p was the root of the tree
  root=rp;
 else
  if(p==q->left)
   q->left=rp;
  else
   q->right=rp;
 delete(p);
}
void print(nodeptr p)
{
 if(p!=NULL)
 {
  print(p->left);
  cout<<p->k<<" ";
  print(p->right);
 }
 else
  cout<<". ";
}
```

```
void main()
{
 nodeptr tree;
 int number;
 cin>>number;
 tree=maketree(number);
 while(cin>>number,number!=0)
  build(tree,number);
 print(tree);
 cout<<"\nEnter the number you want search for and delete\t";
 int n;
 cin>>n;
 search(n,tree);
 print(tree);
}
```

أما جزء الخوارزمية الخاص بإضافة عقدة لشجرة البحث الثنائية فيكون كالآتي:

```
Tree-insert(T, z)
        y = NIL
        x = root[T]
        while x ≠ NIL
                do y = x
                        if key[z] < key[x]
                                then x = left[x]
                                else x = right[x]
        p[z] ← y
        if y = NIL
                then root[T] ← z
                else if key[z] < key[y]
                        then left[y] ← z
                        else right[y] ← z
```

y is maintained as the parent of x, since x eventually becomes NIL.

The final test establishes whether the NIL was a left or right turn from y.

5-3: تعقيدات خوارزمية البحث (Complexy Of Algorithm Search):

- إن تعقيد الخوارزمية الزمني في الحالات الأكثر تعقيدا يُمكن من تحديد الحد الأقصى لعدد العمليات التي يجب استعمالها لترتيب عناصر مجموعة مكونة من n عنصر ،حيث نستخدم الصيغ التقاربية لوصف هذه التعقيدات والتي مر ذكرها في الفصل الأول .
- تعقيد الخوارزمية الزمني في الحالة المتوسطة يُمكن من مقارنة خوارزميات الترتيب و إعطاء فكرة عن الوقت اللازم لتنفيذ الخوارزمية.
- تعقيد الخوارزمية المكاني في الحالات الأكثر تعقيدا أو الحالات المتوسطة يُمثل كمية الذاكرة المستعملة في خوارزمية الترتيب و هي أيضا مرتبطة بعدد عناصر المجموعة.

$$T(n) = O(nlog(n))$$

ولكي نفهم التعقيدات الخاصة بخوارزمية معينة يجب أن نوضح عدة مفاهيم هي:

1- العملية الأساسية (Basic Operation) :

عند دراسة تعقيد خوارزمية نركز على العملية الأساسية لهذه الخوارزمية مثال ذلك في خوارزميات البحث العملية الأساسية هي عملية المقارنة بين القيمة التي يتم البحث عنها وقيم مجموعة البحث، فكلما كان عدد عمليات المقارنة أقل كانت الخوارزمية أكثر فعالية.

مثال// لاحظ تعقيدات جزء البرنامج الآتي :

```
int sum = 0;
for ( int i = 1; i <= n; i++ )
sum += i;
```

نلاحظ أن كل من العمليات sum = 0, i = 1 تنفذ مرة واحدة ،
أما العمليات (i <= n, i++, sum+=i) فتنفذ n مرة ،وبالتالي تابع التعقيد لهذه الخوارزمية هو :

$$f(n) = 3n + 2$$

نقول عن هذه الخوارزمية أنها ذات تعقيد خطي(Linear Complexity) لأن تابع تعقيدها من الدرجة الأولى.

2- التعقيد المقارب (Asymptotic Complexity):

بعد إيجاد تابع التعقيد لخوارزمية ما، وعلى اعتبار عدد المعطيات n كبيراً نهمل الحدود الصغيرة ونهتم فقط بالحد الأعلى، كما نهمل الثوابت الضربية للحد الأعلى.
فإذا كان تابع التعقيد للخوارزمية الآتية :

$$f(n) = 7n^3 + 5n^2 + n + 10$$

نهمل الثابت 7 كما نهمل الحدود 5 ,10 ,n ,n^2 أمام n^3 فيصبح تابع التعقيد المقارب لهذه الخوارزمية هو

$$f(n) = n^3$$

3- التعقيد الزمني في الحالات الأفضل والأسوأ والمتوسطة للخوارزميات
:(Beast & West &Average Complexiy)

في المثال السابق تابع التعقيد هو نفسه دائماً، ولكن في معظم المسائل عدد العمليات لا يتعلق فقط بقيمة الحجم (n) بل يتعلق أيضاً بمحتوى البيانات المعالجة فمثلاً سنأخذ حالة البحث التسلسلي عن عنصر في قائمة من العناصر حيث تتم في هذه الخوارزمية مقارنة العنصر x الذي نبحث عنه مع سلسلة من الأعداد المخزنة في مصفوفة d وطول السلسلة n ، وإرجاع دليل العنصر في حال العثور عليه أو إرجاع القيمة 1- في حال عدم العثور عليه.

مثال// جزء برمجي يوضح تعقيدات بحث عن عنصر x في قائمة معينة d[].

```
int j = 1;
  while (j <= n  && d[j] !=  x)
    j ++;
  if (j > n)
    return -1;
  return j;
```

مثال // برنامج يبين مفهوم التعقيد Complexity علما أن الشكل الأسهل على الفهم في البرامج (قد لا تشمل جميع قواعد لغة ما) .

```
{
        int mid;
        int first = 0;
        int last = N -1;
        while ( first <= last )
        {
                mid = (first + last) / 2;
                if ( list[mid] == target )
                        return mid;
                if ( list[mid] > target )
                        last = mid - 1;
                else    first = mid + 1;
        }

}
```

أن تعقيدات الحالة الأفضل نحصل عليها عندما يكون العنصر المطلوب إيجاده هو نفسه عنصر المنتصف، وبالتالي نحتاج لعملية مقارنة وحيدة.

أما التعقيدات في الحالة الأسوأ فإنها قد تحتاج لعلاقة رياضية لإيجادها فلو كان لدينا مجموعة من الأسماء موجودة ضمن سلسلة ما كما في الشكل، وأردنا البحث عن اسم "جوزيف"، فإننا سنجده في ثلاث خطوات باستخدام خوارزمية البحث الثنائي، بينما سنكون بحاجة إلى 7 خطوات لإيجاده باستخدام البحث الخطي.

```
  begin                mid                           end
    0       1      2      3       4      5      6       7
 ]  هشام    علاء   بكري   ميشيل   أحمد   فوزي   جوزيف   حسن  [
                        begin          mid             end
                                       begin   mid     end
                                               begin   end
```

وسيكون عدد المرات التي ندخل فيها الحلقة while متناسباً مع عدد المرات التي يجب علينا فيها تقسيم السلسلة ذات الحجم N على 2 ، وذلك حتى الوصول لسلسلة مؤلفة من عنصر واحد نبدأ وتنتهي بهذا العنصر، وهو العنصر المطلوب.

أن العلاقة لوغاريتمية من الأساس 2 (لوغاريتم ثنائي) طالما أننا دوماً نقسّم السلسة الناتجة على 2 للحصول على سلسلة (مستوى) جديد.
وبالتالي فإن التعقيد في أسوأ الأحوال هو أن نستمر في التقسيم على 2 حتى يظهر العنصر المطلوب في آخر خطوة تقسيم أو لا يظهر أبداً ، وبالتالي سنحتاج إلى $\log_2 n$ خطوة كأسوأ تعقيد وفي مثالنا سيكون 3 خطوات مع وجود 8 عناصر.

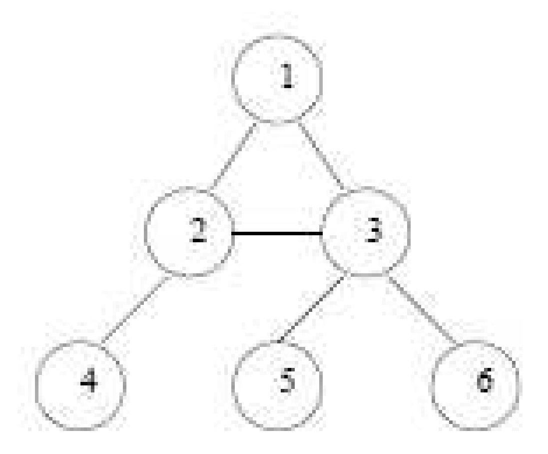
أما بالنسبة للتعقيد الوسطي فانه لا يمكن الوصول لصيغة نهائية فاحتمال وجود العنصر في مكان ما في السلسلة هو احتمال وجود العنصر في السلسة الأصلية مضروباً باحتمال وجوده في السلسلة الفرعية الثانية مضروباً باحتمال وجوده في السلسلة الفرعية التي بعدها وهكذا.
بالتأكيد نجد أن البحث الثنائي أفضل بكثير من البحث الخطي وذلك لاختصاره عدد كبير جداً من المقارنات في حال كانت n كبيرة بشكل كافٍ..مثلاً عندما $n=100000$ فإن أسوأ تعقيد نحصل عليه في حالة البحث الخطي هو $n=100000$.

إما في حال البحث الثنائي فإن أسوأ تعقيد سيكون $\mathrm{Log}_2(100000)=16$ ، أي أن البحث الثنائي وفّر لنا 999,984 عملية مقارنة نسبةً للبحث الخطي.

الفصل الرابع
الامثلية في مسائل تصميم الخوارزميات
Optimization in) Algorithms Design (Equations

4-1: المخططات (Graphs):

المخطط عبارة عن مجموعة من العناصر التي تمثل بنقاط (رؤوس) تسمى (Vertices) وهذه العناصر ترتبط بعلاقات تسمى حافات (Edges) وهذه العلاقات تمثل بخطوط كما في شكل رقم (16) التالي :



شكل رقم (16) يوضح مخطط غير متجه بسيط

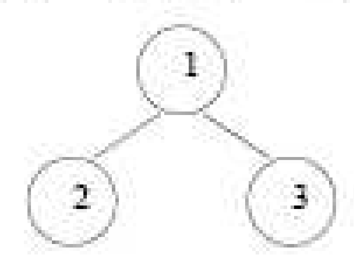
G =(V,E)
V(G) = {1,2,3,4,5,6}
E(G) = {(1,2),(2,1),(1,3),(3,1),(2,3),(3,2),(2,4),(4,2),(3,5),(5,3),(3,6),(6,3)}

حيث إن (V) تمثل مجموعة من النقاط و(E) تمثل مجموعة من الحافات.

4-3: أنواع المخططات (Type Of Graphs) :

يوجد نوعين من المخططات هي :

1- المخطط غير المتجه (Un directed Graph) : هو المخطط الذي تكون العلاقة بين عناصره غير مرتبة أي إن الاتجاهات تكون غير مهمة كما في الشكل رقم (17) .

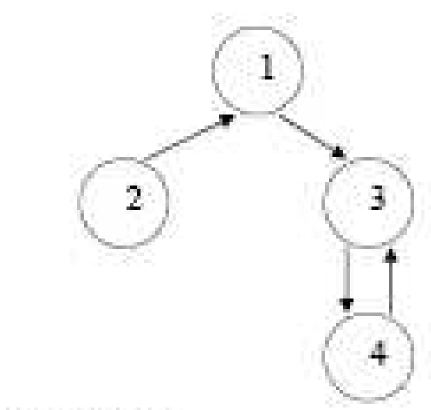


V(G) = {1,2,3}
E(G) = { (1,2),(2,1),(1,3),(3,1)}

شكل رقم (17) مخطط غير متجه

2- المخطط المتجه (Directed Graph):

هو المخطط الذي تكون العلاقة بين عناصره مرتبة بنمط معين أي إن الاتجاهات تكون مهمة ومحسوبة كما في الشكل رقم (18).



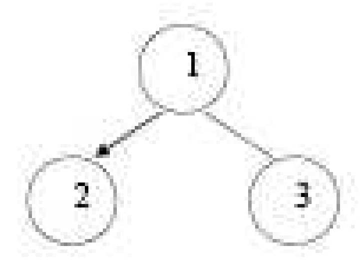
V(G) = {1,2,3,4}

E(G) = { (1,2),(3,1),(3,4),(4,3)}

شكل رقم (18) مخطط متجه

2- المخطط المشترك (Un directed Graph & directed Graph):

هو المخطط الذي يحوي كلا النوعين كما في الشكل رقم (19) التالي :



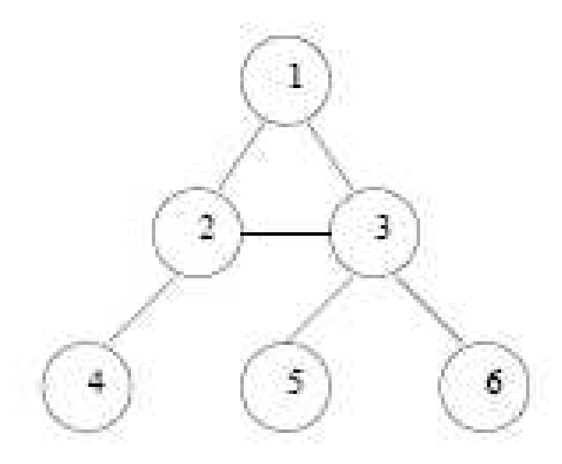
V(G) = {1,2,3}

E(G) = { (2,1),(1,3),(3,1)}

شكل رقم (19) مخطط مشترك

المسار : هو مجموعة من المستقيمات التي تربط بين نقطتين في المخطط .

مثال// في الشكل رقم (20) التالي اوجد المسار بين النقاط التالية (1,5) ، (1,6) ؟



شكل رقم (20) يوضح مخطط يحوي مجموعة مسارات

الحل //

1. المسار الأفضل للنقطة (1,5) هو: (3,5)، (1,3) وليس (3,5) ،(2,3)،(1,2)
2. المسار الأفضل للنقطة (1,6) هو: (3,6)، (1,3) وليس (3,6) ،(2,3)،(1,2)

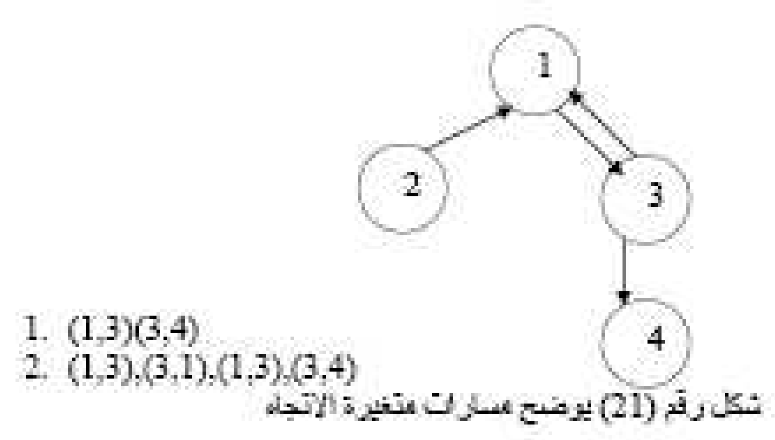
مع ملاحظة انه لا يمكن كتابة المسار باستخدام أقواس المجموعه { }.

3-4: طول المسار (Path Length) :

هو عدد المستقيمات (الخطوط) التي تربط نقطتين في المخطط ،كما نلاحظ في مخطط المثال أعلاه :

أ- (1,5) طول المسار بينهما هو (2) ويمكن إن نأخذ الطول الآخر وهو (3)
ب- (1,6) طول المسار بينهما هو (3) ويمكن إن نأخذ الطول الآخر وهو (2)

ولمعرفة طول المسار فانه علينا إن نحسب النقاط أي نحسب عدد الأزواج (عدد المستقيمات). في المخططات المتجه أحيانا يوجد أكثر من مسار بين نقطتين وبالتالي فانه لدينا مشكلة تكرار في طول المسار (مسار متغير) كما في الشكل رقم (21) التالي:



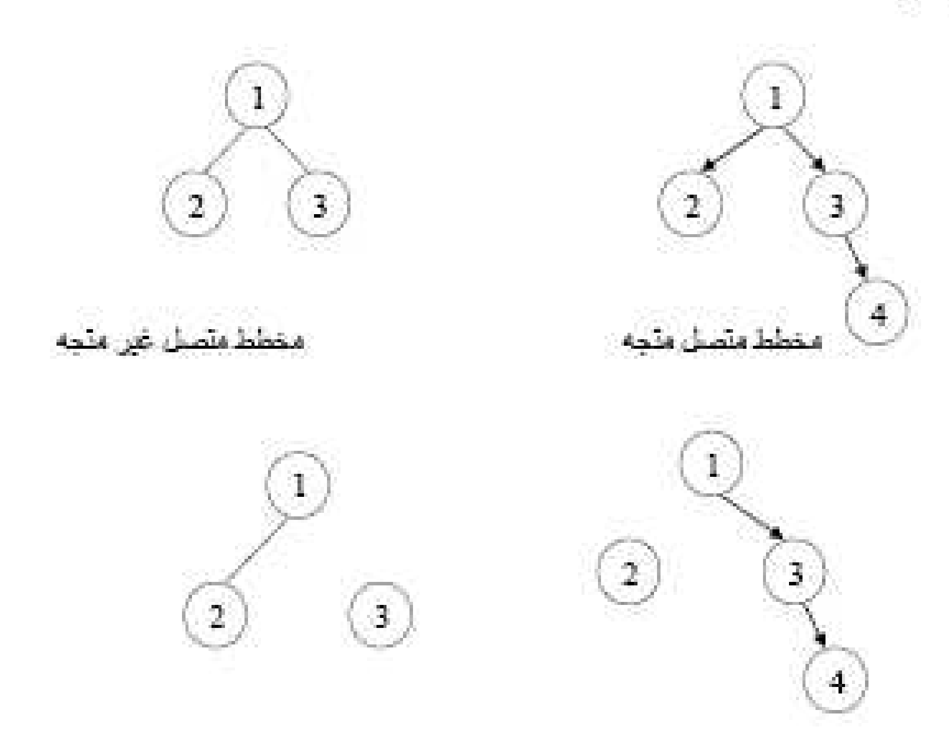
شكل رقم (21) يوضح مسارات متغيرة الاتجاه

المخطط المتصل (Connected Graph) :

هو المخطط الذي توجد فية مسارات بين أية نقطتين من نقاط المخطط

المخطط غير المتصل (Un Connected Graph) :

هو المخطط الذي تكون بعض نقاطه غير متصلة فيما بينها بمسار ،والشكل رقم (22) التالي يوضح ذلك:

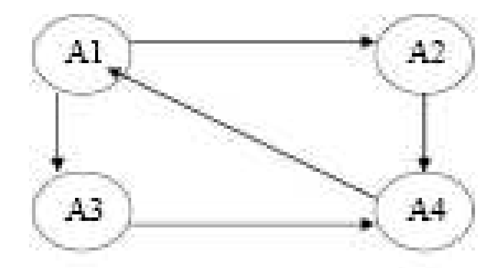


مخطط غير متصل متجه | مخطط متصل غير متجه

شكل رقم (23) يوضح أنواع المخططات

يمكننا استخدام المخططات لتحديد أو لمعرفة اقصر المسارات من خلال إيجاد كل المسارات ومن ثم بيان الأفضل فيها .

مثال// لديك المخطط التالي :



المطلوب //
1. توضيح نوع المخطط
2. تمثيل المخطط بمصفوفة
3. بيان قيم المصفوفة
4. بيان القيم الداخلة والخارجة من كل نقطة
5. إيجاد اقصر مسار بين نقطة A1 والنقطة A4

الحل//
1. المخطط متجه ومتصل (Direct Graph & Connected Graph).
2. يمكن تمثيل المخطط بالمصفوفة التالية :

	1	2	3	4
1	A11	A12	A13	A14
2	A21	A22	A23	A24
3	A31	A32	A33	A34
4	A41	A42	A43	A44

3. يمكن بيان قيم المصفوفة أعلاه كالتالي:

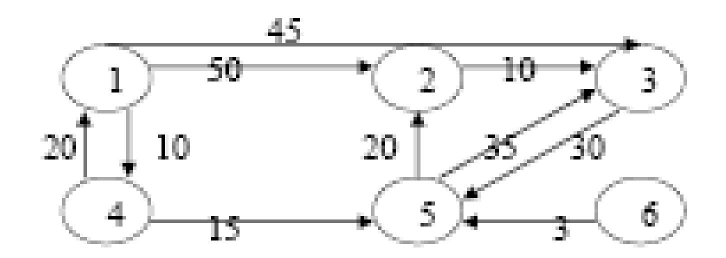
		A1	A2	A3	A4
2	A1	0	1	1	0
1	A2	0	0	0	1
1	A3	0	0	0	1
1	A4	1	0	0	0
		1	1	1	2

4. لإيجاد مجموع القيم الداخلة والخارجة من كل نقطة فان :
مجموع القيم في كل صف يمثل عدد الخطوط الخارجة عن كل نقطة
(Row = Out degree)
مجموع القيم في كل عمود يمثل عدد الخطوط الداخلة للنقطة
(Column = Input degree)

اسم النقطة	المسارات الخارجية	المسارات الداخلية
A1	2	1
A2	1	1
A3	1	1
A4	1	2

5. لإيجاد اقصر مسار بين نقطة A1 والنقطة A4
a. المسار (A1,A2)، (A2,A4)
b. المسار (A1,A3)، (A3,A4)
وهما أفضل المسارات لان درجتهما تساوي (2)

مثال // لديك المخطط التالي :



المطلوب//

1- حدد نوع المخطط
2- حدد المصفوفة
3- حدد قيم المصفوفة
4- حدد القيم الداخلة والخارجة
5- حدد المسارات بين نقاطه
6- أعط اقصر مسار ما بين النقطة (1,6) و(1,5) و(2,6).

الحل//

1. المخطط متصل ومتجه
2. المصفوفة هي :

	1	2	3	4	5	6
1	A11	A12	A13	A14	A15	A16
2	A21	A22	A23	A24	A25	A26
3	A31	A32	A33	A34	A35	A36
4	A41	A42	A43	A44	A45	A46
5	A51	A52	A53	A54	A55	A56
6	A61	A62	A63	A64	A65	A66

3. تحدد قيم المصفوفة كالتالي:

	A1	A2	A3	A4	A5	A6
A1	0	50	45	10	0	0
A2	0	0	10	0	0	0
A3	0	0	0	0	30	0
A4	20	0	0	0	15	0
A5	0	20	35	0	0	0
A6	0	0	0	0	3	0

4. لتحديد المسارات الداخلة والخارجة:

المسارات الداخلة	المسارات الخارجة	اسم النقطة
20	105	A1
70	10	A2
90	30	A3
10	35	A4
48	55	A5
0	3	A6

5. لإيجاد أقصر مسار بين النقاط كالتالي:

a. المسار بين (1,6) هو : لا يوجد مسار بينهما
b. المسار بين (1,5) هو : (4,5) ،(1,4)
c. المسار بين (2,6) هو : لا يمكن الوصول إلى النقطة (6) لعدم وجود أي مسار داخل إليها .

4- 4: طريقة الجموح أو الطماع (Greedy Method):

إن هذه الطريقة تستخدم غالباً لحل مسائل الأمثلية (Optimization problems)التي عادة إما تكبر (Maximum) لشيء معين أو تصغر (Minimum) قدر الإمكان لنفس الشيء ، كما في حالة الربح أو الخسارة .

إن هذه المسائل تحتوي على العناصر التالية:

أ- دالة هدف (Objective Function) وإن الحل لها يجب إن يحقق قيود معينة للمسألة ويكون حل ممكن ويسمى أفضل الحلول الممكنة وهو الأفضل .

ب- إن مجموعة القيود هذه تسمى (Constraints) ،وإن مجموعة الحلول التي تحقق القيود تسمى حلولاً ممكنة (Feasible Solutions) والحل الممكن الذي يعطي أفضل دالة هدف يسمى الحل الأمثل (Optimal Solution) .

إن الفكرة لهذه الطريقة تتمثل في :

(يبنى الحل الأمثل في طريقة الجموح على مراحل ،في كل مرحلة يُتخذ أفضل قرار تبعاً لمقياس أمثلية مناسبة ، وحيث إن القرار المتخذ لن يتغير لاحقاً لذا يجب إن يحقق أمثلية).

ملاحظة// إن اغلب المسائل التي تحلها طريقة الجموح تتكون من أكثر من واحد من المداخل (n input) . حيث دائماً ينتقي واحد من المداخل الذي حالياً هو الأمثل ولكن قد يكون أسوأ لاحقاً لذا فهذا يعني انه يوجد مساوئ أو مشاكل لهذه الطريقة .

يوجد نموذجين لطريقة الجموح هما:

1. نموذج المجموعة الجزئية (Subset Paradigm):

في هذا النموذج يتم انتقاء مجموعة جزئية مثلى من المدخلات تبعاً لمقاييس أمثلية معينة ويجب إن تحقق هذه المجموعة قيود المسألة ، وفيما يلي تجريد ضبطي أو تحكمي لهذا النموذج:

```
SolType   Greedy (type  a[],int n)
//a[1..n] contains the n inputs.
{soltype .solution=Empty;//initialize the solution .
 For(int i=1;i<=n;i++)
{ typeX=Select(a);
   If  feasible(solution ,X)
         Solution=Union(solution,X);
   Return  solution;
```

إن المصفوفة (Solution) هي مصفوفة أحادية البعد وهي لا تحوي أي عنصر في البداية ،إما (Select) فهي دالة تدخل لها مجموعة مدخلات وترجع واحد هو الأمثل ثم يتم الاختيار هل إن الحل هو حل ممكن فإذا نعم فانه يضيف هذا الحل لمجموعة الحلول السابقة وفي حالة العكس فانه يبعد عن الحل الأمثل .

5-4: مسألة الجراب (Knapsack Problem) :

سنأخذ مسألة الجراب أو حقيبة الظهر كمثال لنموذج المجموعة الجزئية (Knapsack Problem) : حيث هناك مجموعة كيانات توضع في هذه الحقيبة .

معطيات المسألة : (n) من الكيانات حيث ممكن إن تكون أي شيء ولدينا جراب سعته (c) مقاساً بالكيلو غرام (لاحظ كم سيتحمل وزن من هذه الكيانات)
للكيان (i) الوزن (W_i) حيث ($1 \le i \le n$)
إضافة الكسر (Xi) وهو يمثل حل المسألة حيث ($0 \le Xi \le 1$) يحقق فائدة هي ($PiXi$) .
المطلوب : ملئ الجراب بحيث تعظم الفائدة الحاصلة أي انه هناك مسألة (Maximum)
دالة الهدف هي معيار الأمثلية (القيمة الكبر لهذه الدالة هي معيار الأمثلية أي الحل الأمثل)

$$Maximise \quad \sum_{i=1}^{n} PiXi$$

ثم طباعة المتجه (X) حيث بالاعتماد على قيمة المتجه فانه ممكن جزء من الفائدة يضاف أو لا يضاف .
إن قيود المسألة يجب إن تحقق :

$$\sum_{i=1}^{n} WiXi \quad \le \quad C$$

1. إن أي حل يحقق قيود المسألة هو حل ممكن.
2. أي حل يحقق اكبر تغير بدالة الهدف هو الحل الأمثل .

بالإضافة إلى إن :

$0 \le Xi \le 1 \ ,1 \le i \le n$

and

$Pi \ \phi \ 0, Wi \ \phi \ 0,1 \le i \le n$

وسنأخذ الآن مثالاً تطبيقياً لهذه الطريقة :

$P[1..3] = (25, 24, 15), \ C = 20, \ n = 3$

$W[1..3] = (18, 15, 10), \ \frac{Pi}{Wi} (1.3, 1.6, 1.5)$

إن هذه الطريقة تعطي ثلاث حلول ممكنة ،نختار أفضلها كالآتي:
الحل الأول : انه أول حل نختاره يكون هو الحل الأكبر بالفائدة .
الحل الثاني : انه أول حل نختاره يكون هو الحل الأكبر بالوزن .
الحل الثالث : انه أول حل نختاره يكون بالاعتماد على نسبة تقسيم الفائدة على الوحدة المختارة .
سوف نقوم بعمل جدول لتوضيح حلول المسألة:

$(x1, x2, x3)$	$\sum WiPi$	$\sum PiWi$	المقياس
(1,2/15,0)	20	28.2	الفائدة الأعلى أولا
(0,2/3,1)	20	31	الوزن الأقل أولا
(0,1,1/2)	20	31.5	الفائدة الأكبر لكل وحدة وزن أي $(\frac{Pi}{Wi})$ أولا

ولتوضيح هذه النتائج نطبق الخيار الأول كالتالي :

20-18 =2
2-15 =2/15
0-10 =0/10 = 0

هنا الكمية (18) تعتبر كمية واحدة وتساوي (1) حيث قمنا بطرح الوزن من الجراب وهكذا نستمر بهذه العملية حتى يتم ملئ الجراب .
مع ملاحظة النتائج للخيارات أعلاه نلاحظ إن المقياس الثالث هو الذي أعطى نتائج اكبر فائدة وبالتالي فان قيم (x1,x2,x3) التابعة له تمثل الحل الأمثل.

وللتأكد من صحة النتائج يمكننا إجراء عملية ضرب بين قيم المتجهات الناتجة والأوزان الخاصة بها .

وهذا هو الإجراء التنفيذي الذي يقوم بحل مسألة Knapsack Problem

```
Void Greedy Knapsack (float m, int n)
// p[1..n] and w[1..n] contain profits and weights.
// respectively of the n-objects ordered such
that p[i]/w[i] ≥ p[i+1]/w[i+1].
//m is the Knapsack size and x[1..n] is the solution vector.
```

إن (n) هنا تمثل عدد الكيانات و(m) تمثل سعة الجراب ، وجميع الكيانات المرتبة تعتمد على هذه النسب . حيث يصبح الترتيب تنازليا إثناء الحل وبالتالي فانه دائما يكون الانتقاء للحل الأول والذي يمثل الأمثل،وفي البداية فان متجه X يكون فارغ تماماً أي يحوي قيماً صفرية

```
{
 For(int i=1;i<=n;i++)     x[i]=0.0;
 Float U=m;
 For(i=1;i<=n;i++)
  {
    If (W[i] > U)  break;
       X[i]=1.0;
       U- = W[i];
   }
  If (i<=n)     x[i]=U/W[i];
}
```

تعقيدات الوقت للدالة :

تتطلب هذه الخوارزمية بغض النظر عن وقت ترتيب الكيانات ابتدائياً $O(n)$ من الوقت فقط حيث إننا سنقيس تعقيدات الوقت بدون حساب وقت الترتيب، حيث انه في حالة استخدام خوارزمية ترتيب الدمج فان وقت التعقيد لها هو $nLogn$ لأنها اكبر دائماً وتعطي حلاً امثل.

ملاحظة// يوجد خوارزمية تسمى (0/1 Knapsack) حيث أنها إما تحمل القيمة أو لاتحملها وبالتالي فانه يجب حذف تعليمة (If) الأخيرة من هذه الخوارزمية وفي هذه الحالة لا يكون هناك ضمان للحصول على حل امثل .

2. نموذج الترتيب (Ordering paradigm):

في هذا النموذج يتم اتخاذ القرارات باعتبار المدخلات بترتيب معين ,حيث كل قرار يتم اتخاذه باستخدام معيار للامثلية والذي يمكن حسابه من خلال القرارات المتخذة سابقاً .

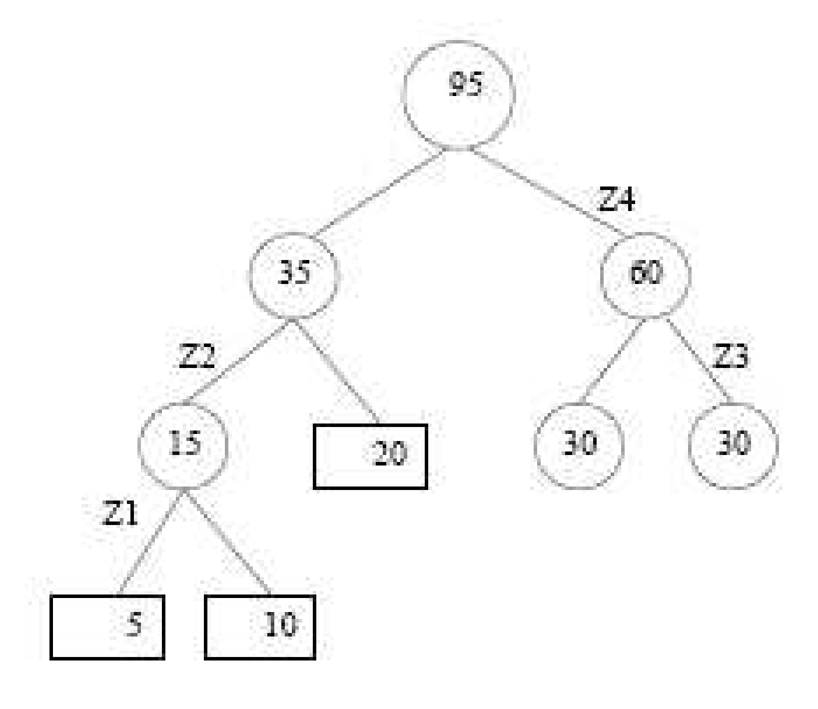
ملاحظة// في مسائل التباديل ممكن تبديل واحد يحصل يؤدي إلى تغير الحل .

لتوضيح هذا النموذج سوف نأخذ مسألة تسمى مسألة أنماط الدمج الأمثل (**Optimal Merge Patterns**) حيث تتلخص هذه المسألة بالمفاهيم التالية :

يوجد لدينا مجموعة ملفات سنحاول دمجها ليكون لدينا ملف واحد امثل وبأقل تعقيدات وبأقل تحريك للقيود الخاصة بالملفات المدمجة أي انه ستكون مسألة (Minimum).
إن هذه المسألة هي مسألة دمج (n) من الملفات المرتبة على شكل أزواج وتوليد ملف واحد مرتب بأقل عدد ممكن من الحركات للسجلات ، لان هذه المسألة تستدعي الترتيب ما بين أزواج من الملفات المراد دمجها لذلك فهي تطابق نموذج الترتيب.

إن قاعدة الجموح هي :
(لتقليص حركة السجلات ادمج الملفين الأقل حجماً عند كل خطوة أولا).

مثال// لديك مجموعة الفايلات الموضحة كما في التالي:
[F1..F5] = (20,30,10,5,30) ,n = 5



إن القيمة (20) تمثل عدد القيود أو السجلات في الملف الأول، وهكذا بالنسبة للبقية أي إن القيمة (30) تمثل عدد القيود في الملف الثاني .
إن القيمة (10) تمثل طول الملف .
إن ترتيب الدمج لهذه الملفات هو الآتي:

20,30,15,30
30,30,35
60,35

إذا كانت d_i تمثل بعد العقدة الخارجية للملف F_i فإن q_i يمثل طول الملف F_i
أي إن العدد الكلي لحركات السجلات لشجرة الدمج الثنائية هذه تكون :

$$\sum_{i=1}^{n} d_i q_i$$

$= 5*3+10*3+20*2+30*2+30*2$
$=205$

وهذا هو الحل الأمثل الذي يكون هو الحجم الأقل دائماً .
تمرين: تطبيق عملية دمج عشوائي ؟

وهذه هي الدالة التي تولد الشجرة الثنائية مكتوبة بلغة ++C :

```
Struct Tree node
{ Struct tree*Lchild,*Rchild;
  Int  Weight ;
}
```

إن الجزء أعلاه خاص بشكل العقدة للشجرة الثنائية بالقائمة الموصولة

```
Typedef   Struct Tree node type;

Type *tree (int n)
//List is global list of n single node binary trees as described above.
```

n تمثل عدد الملفات ،وهذا تمثيل للقائمة ،w تمثل عدد القيود لكل ملف

0	W	0	....		n	

```
For (int i=1;i<n;i++)
{ type *pt=new type;
Pt->Lchild=Least (list);
Pt->Rchild=Least (list);
Pt->Weight=(pt->Lchild)->Weight+ (pt->Rchild)->Weight;
Insert (list,*pt);
}
```

إن هذه الـ for تكون خاصة بعملية الدمج بين الملفات ،الدالة Least ترجع مؤشر إلى العقدة التي تكون ذات اقل وزن من الملفات وتحذف مكانها ،الدالة insert تقوم بإضافة عنصر إلى القائمة list الخاصة بالعقد ، pt* موقع محتويات المؤشر ،

```
Return (Least(list));
}
```

إن الناتج من هذه العملية هو مؤشر إلى شجرة الدمج الثنائية .

وفيما يلي توضيح موجز للخوارزمية :

1. كل شجرة في القائمة List تمتلك بالضبط عقدة واحدة ، هذه العقدة هي عقدة خارجية حيث تتكون من ثلاث حقول هي : (Rchild)، (Weight)، (Lchild).
(Rchild) و(Lchild) يحتويان قيماً صفرية ، إما الحقل (Weight) فيحتوي على طول احد الملفات المراد دمجها .
2. الدالة (Tree) تستعمل دالتين اخريتين هما (Least) و(Insert) ،حيث إن الدالة (Least) تقوم بإيجاد شجرة في القائمة جذرها يمتلك الوزن الأقل وتعيد هذه الدالة مؤشر إلى هذه الشجرة وتقوم بعد ذلك بحذفها من القائمة .
إما الدالة (Insert) فإنها تقوم بحشر الشجرة التي جذرها Pt إلى القائمة (List) حيث إن مؤشر هذه العقدة هو Pt .

3. إن شجرة الدمج الثنائية الناتجة في نهاية هذه الخوارزمية تستخدم لتحديد أية ملفات يتم دمجها حيث يُنجز الدمج على تلك الملفات التي تمتلك العمق الاكبر في الشجرة .

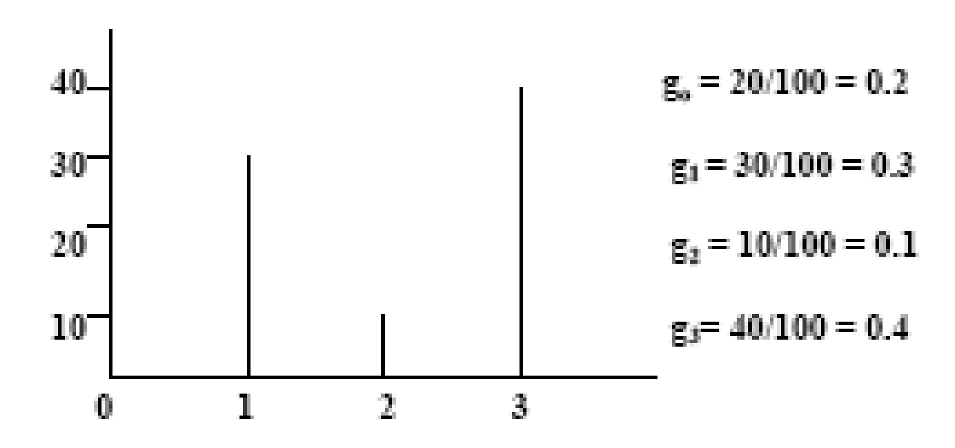
تعقيدات الوقت للخوارزمية :

إن حلقة for الرئيسية تتكرر (n-1) من المرات ، في حالة الاحتفاظ بالقائمة (List) مرتبة تصاعدياً نسبة إلى قيم الأوزان في الجذور فان الدالة (Least) تتطلب (O(1)) من الوقت والدالة (Insert) ممكن انجازها بوقت هو ($O(n)$) ،أي إن الوقت الكلي المستغرق هو ($O(n^2)$) .

6-4: استخدام قاعدة الطماع في إيجاد أمثلية البيانات :

طريقة هوفمان (Huffman code) سميت نسبة للعالم هوفمان 1952 ، تعتمد على فكرة تقليل البيانات وضغطها بدون التأثير على دقة المعلومات أي بدون فقدان البيانات

مثال// لديك البيانات التالية استخدم طريقة هوفمان لقاعدة للجموح Greedy rule ممثلة بالمدرج التكراري التالي



أ- تمثيل قيم المدرج

$g_3 \rightarrow 0.4$ 0.4 → 0.4
$g_1 \rightarrow 0.3$ 0.3 → 0.3
$g_0 \rightarrow 0.2$ 0.2 → 0.3
$g_2 \rightarrow 0.1$ 0.1 ┘

ب- ترتيب القيم وجمع اقل قيمتين

0.4 → 0.4 → 0.4 0.4 → 0.4 → 0.6
0.3 → 0.3 → 0.6 0.3 → 0.3 → 0.4
0.2 → 0.3 ┘ 0.2 → 0.3 ┘
0.1 ┘ 0.1 ┘

ج- الاستمرار بجمع القيم الصغيرة لحين الوصول لقيمتين فقط

شفرة الاعتيادية **Original gray level (natural code)**	الاحتمالية **Probability**	شفرة هوفمان **Huffman code**
g_0 : 00_2	0.2	010_2
g_1 : 01_2	0.3	00_2
g_2 : 10_2	0.2	011_2
g_3 : 112	0.4	1_2

إيجاد **Entropy** الخاص بالاحتمالية المستخدمة:

$$\text{Entropy} = -\sum_{i=0}^{3} P_i \log_2 (P_i)$$
$$= -[(0.2)\log_2(0.2) + (0.3)\log_2(0.3) + (0.1)\log_2(0.1) + (0.4)\log_2(0.4)] \approx 1.846 \text{ bits/pixel}$$

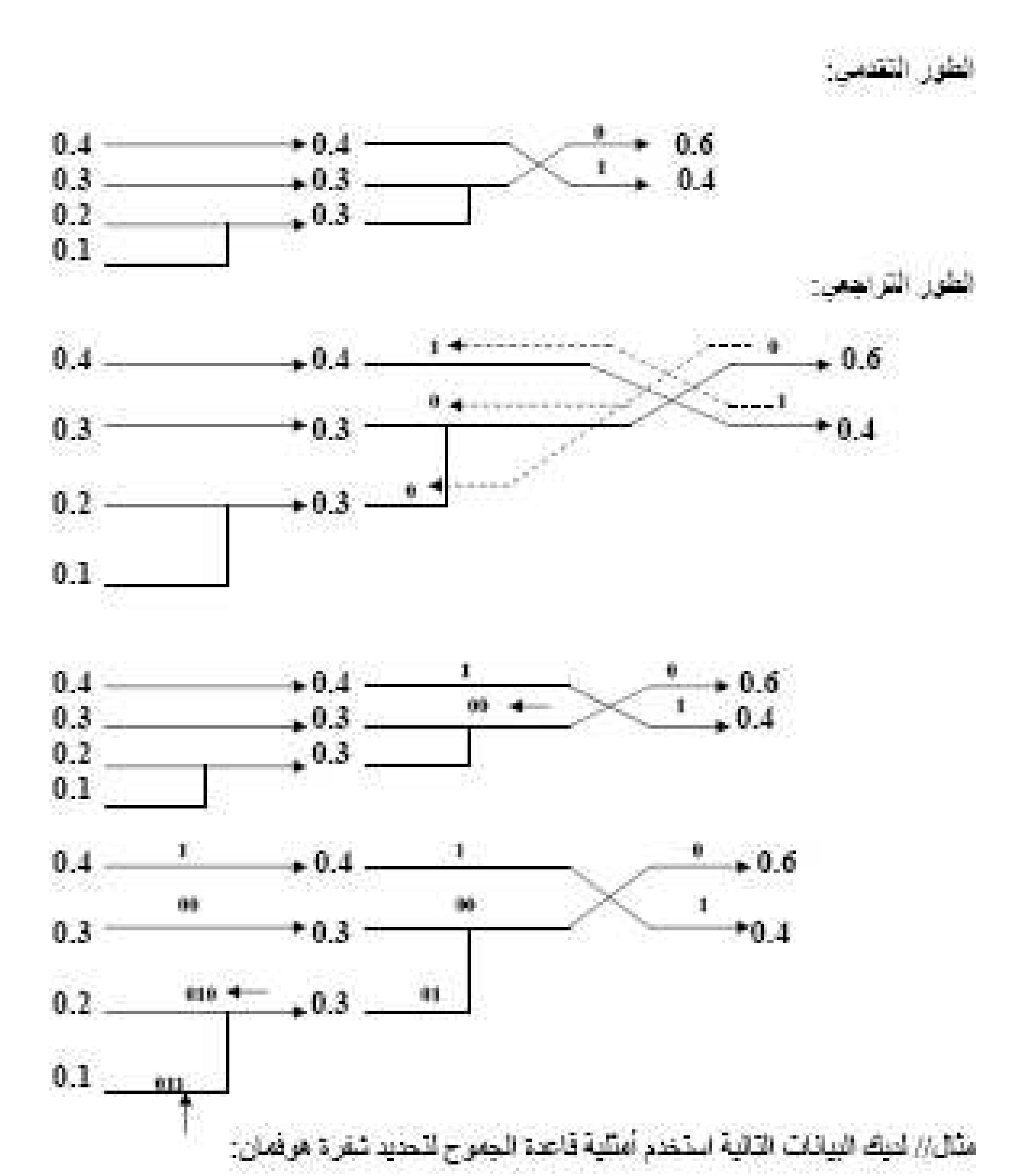
علما إن إيجاد $\log_2(X)$ يمكن الحصول علية حسب القانون التالي:

$$\log_2(X) = 3.322 * \log_{10}(X)$$

إيجاد معدل طول الشفرة حسب القانون التالي (**Average length**):

$$L_{ave} = \sum_{i=0}^{L-1} L_i P_i$$

$3(0.2) + 2(0.3) + 3(0.1) + 1(0.4) = 1.9$ bits / pixel



مثال// لديك البيانات التالية استخدم أمثلية قاعدة الجموع لتحديد شفرة هوفمان:

Huffman Code Example (from our text)

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Original source			Source reduction							
Sym.	Prob.	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Although it might not look like it, this is both uniquely decodable and instantaneous code. Think about how you'd decode an incoming bit stream.

<u>Entropy of this example</u>

$$H = -(0.4\log(.4,2) + 0.3\log(.3,2) + 2\cdot 0.1\log(0.1,2) + 0.06\log(.06,2) + 0.04\log(.04,2)$$

$$H = 2.14$$

<u>Average code length</u>

$$R = (.4\cdot 1) + (.3\cdot 2) + (.2\cdot 3) + (.2\cdot 4) + (.06\cdot 5) + (.04\cdot 5)$$

$$R = 2.9$$

note that $H < R$, as expected, but it would be hard (impossible) to find any code which did better (i.e. closer to the optimal entropy H). That is why the Huffman code is a 'compact' code.

شفرة هوفمان مستخدمة الأشجار الثنائية :-

الطريقة المستخدمة تتمثل كالآتي:

1- وجود جملة معينة
2- حساب تكرار كل حرف أو رمز بالجملة
3- عمل شجرة وذلك بجمع اقل قيمتين بكل مرة
4- ترقيم الشجرة بحيث كل مسار على الجهة اليسرى يعطى له (0) وكل مسار على الجهة اليمنى يعطى له (1).
5- كتابة شفرة هوفمان لكل حرف أو رمز بالجملة من خلال تتبع مساره
6- إيجاد معدل الطول والـ Entropy

مثال// اوجد شفرة هوفمان للجملة التالية:

(dead beed cafe deeded dad. Dad faced a faced ab. Dad a cceded , dad be back.)

Space	a	b	c	d	e	f	.
17	12	4	5	19	12	4	4

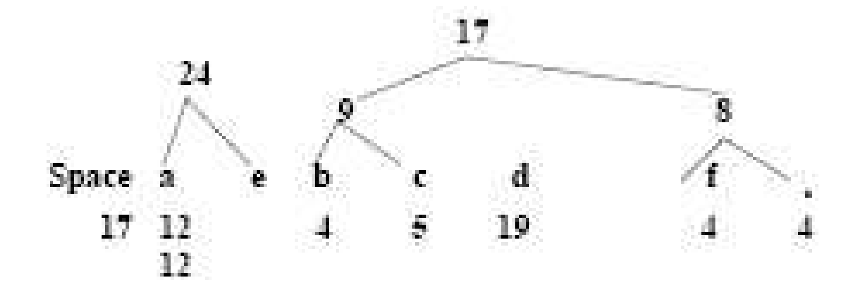
2- جمع اقل قيمتين بكل مرة

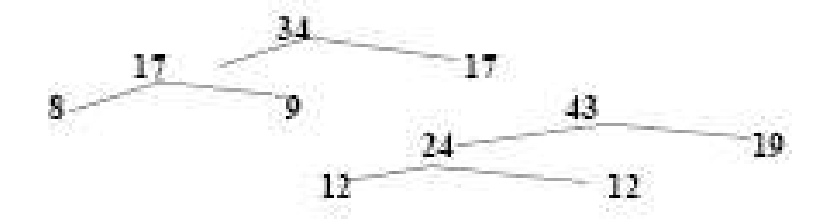
9 8

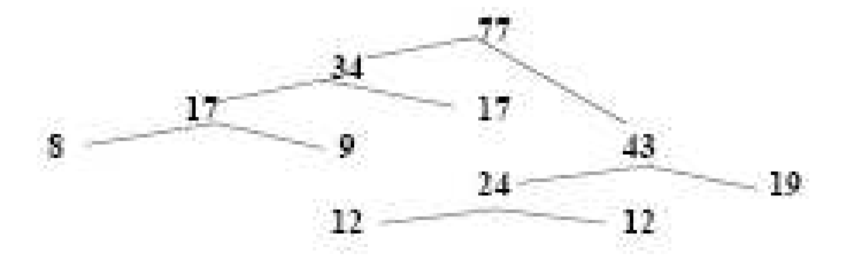
Space	a	b	c	d	e	f	.
17	12	4	5	19	12	4	4

17
9 8

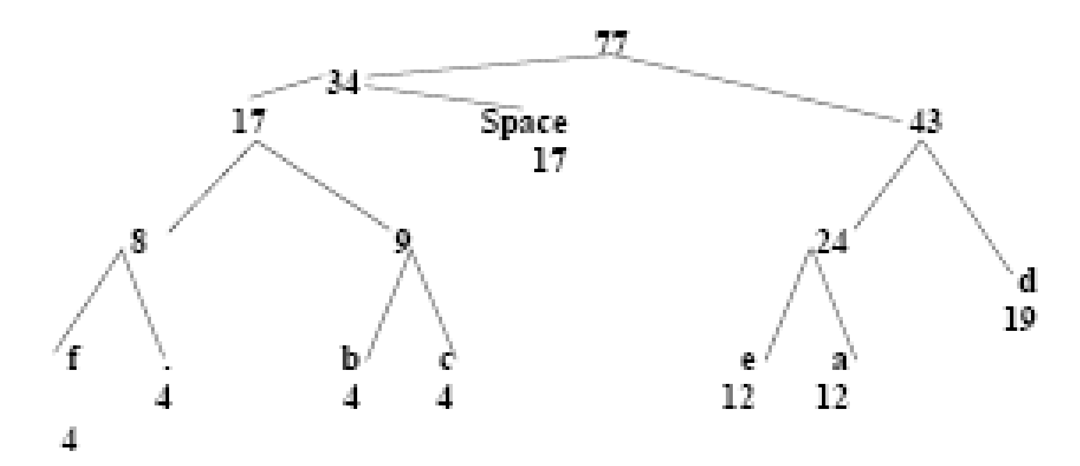
Space	a	b	c	d	e	f	.
17	12	4	5	19	12	4	4







3- ترقيم الشجرة



Space	a	b	c	d	e	f	,	الرمز
01	101	0010	0011	11	101	0000	0001	الشفرة هوفمان

وجزء الخوارزمية الخاص بطريقة شجرة هوفمان هو:

```
Huffman(C)
1.....n = |C|
2.....Q = C
3.....for i = 1 to (n - 1) do
4..........allocate a new node z
5..........z.left = x = Q.Extract-Min
6..........z.right = y = Q.Extract-Min
7..........f[z] = f[x] + f[y]
8..........Insert(Q, z)
9.....return Q.Extract-Min ; return the root of the tree
```

حيث يمكنها تقليل التعقيد من n.log(n) إلى n^2 .

الفصل الخامس
البرمجة الديناميكية
Dynamic programming

1-5: البرمجة الديناميكية (Dynamic programming):

يتم تمثيل نماذج البرمجة الديناميكية بطرق مختلفة أكثر من أي نموذج برمجي رياضي آخر، وعوضا عن استخدام التابع الغرضي والقيود، يصف نموذج البرمجة الديناميكية الإجرائيات من وجهة نظر الحالات، والقرارات، والتحويلات، والمرجعات وهي:

1- يبدأ الإجراء من حالة ابتدائية حيث يتم اتخاذ قرار ما.
2- يسبب هذا القرار الانتقال إلى حالة جديدة.
3- بالاستناد إلى الحالة البدائية، الحالة النهائية والقرار يتم الوصول إلى قيمة مرجعة.
4- يستمر الإجراء عبر سلسلة من الحالات حتى الوصول إلى حالة نهائية.

تكمن المشكلة في إيجاد السلسلة التي تجعل القيمة الكلية المرجعة أعظمية، وتعد نماذج وأساليب البرمجة الديناميكية الأكثر ملائمة للحالات التي ليس من السهل نمذجتها باستخدام قيود البرمجة الرياضية لأنها تقدم فائدة كبرى عندما تكون مجموعة القرارات محدودة، ومتقطعة، وعندما يكون التابع الغرضي غير خطي.
وقد وصفت البرمجة الديناميكية بأنها الطريقة الأعم بين طرق الأمثلة بسبب قدرتها على حل صف واسع من المشاكل، يتبنى هذا النموذج مشاكل معينة بشكل خاص حيث تعبر نفسها إلى إجرائيات حسابية فعالة، أي تلك الحالات التي تتضمن توابع غير مستمرة أو قيم متقطعة، ففي هذه الحالات، قد تشكل البرمجة الديناميكية منهجية الحل الوحيدة الممكنة.

إن الفرق بين قاعدة الجموح(Greedy method) التي تعتمد على معلومات عامة وبين البرمجة الديناميكية (dynamic programming) التي تعتمد على معلومات عالمية هو إن في الأولى يتولد تعاقب قرارات واحدة بينما الثانية يتولد تعاقب قرارات كثيرة لكن القرارات التي تحتوي قرارات جزئية غير مثلى لا يمكن إن تكون مثلى ولهذا لا تأخذ.

فإذا كان لدينا D من الخيارات، لكل قرار فإن هناك n^D تعاقب قرار ممكن.
لذا فإن خوارزميات البرمجة الديناميكية لها تعقيدات متعددة الحدود.

2-5: أمثلة على البرمجة الديناميكية:

مثال// إيجاد الوقت الأفضل لـ n من القيم كالآتي:

- n^a dominates n^b if $a > b$ since

$$\lim_{n\to\infty} n^b/n^a = n^{b-a} \to 0$$

- $n^a + o(n^a)$ doesn't dominate n^a since

$$\lim_{n\to\infty} n^a/(n^a + o(n^a)) \to 1$$

Complexity	10	20	30	40
n	0.00001 sec	0.00002 sec	0.00003 sec	0.00004 sec
n^2	0.0001 sec	0.0004 sec	0.0009 sec	0.016 sec
n^3	0.001 sec	0.008 sec	0.027 sec	0.064 sec
n^5	0.1 sec	3.2 sec	24.3 sec	1.7 min
2^n	0.001 sec	1.0 sec	17.9 min	12.7 days
3^n	0.59 sec	58 min	6.5 years	3855 cent

مثال// يوضح خوارزمية إيجاد اقصر مسافة .

What is $d[i,j]^0$?

$$d[i,j]^0 = 0 \text{ if } i = j$$
$$= \infty \text{ if } i \neq j$$

What if we know $d[i,j]^{m-1}$ for all i,j?

$$d[i,j]^m = \min(d[i,j]^{m-1}, \min(d[i,k]^{m-1} + w[k,j]))$$
$$= \min(d[i,k]^{m-1} + w[k,j]), 1 \leq k \leq i$$

since $w[k,k] = 0$

This gives us a recurrence, which we can evaluate in a bottom up fashion:

```
for i = 1 to n
        for j = 1 to n
        d[i,j]^m = ∞
        for k = 1 to n
                d[i,j]^0=Min( d[i,k]^m, d[i,k]^(m-1) + d[k,j])
```

This is an $O(n^3)$ algorithm just like matrix multiplication, but it only goes from m to $m+1$ edges.

مثال// إيجاد قيمة الوقت لقيمة مرفوعة لأس قيم معينة كما موضح في الجدول الآتي :

n	$f(n) = n$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = n!$
10	0.01 μs	0.1 μs	1 μs	3.63 ms
20	0.02 μs	0.4 μs	1 ms	77.1 years
30	0.03 μs	0.9 μs	1 sec	8.4×10^{15} years
40	0.04 μs	1.6 μs	18.3 min	
50	0.05 μs	2.5 μs	13 days	
100	0.1 μs	10 μs	4×10^{13} years	
1,000	1.00 μs	1 ms		

3-5: تجمع البيانات (Data clustering):

تجمع البيانات هي عملية وضع البيانات في مجموعات مماثلة، خوارزمية التجمع تقسم مجموعة من البيانات إلى عدة مجموعات، حيث أن التشابه بين النقاط ضمن مجموعة معينة أكبر من التشابه بين نقطتين ضمن مجموعتين مختلفتين، إن فكره تجمع البيانات هي فكره بسيطة في طبيعتها وهي قريبه جدا من الإنسان في طريقه تفكيره حيث أننا كلما تعاملنا مع كميه كبيره من البيانات نميل إلى تلخيص الكم الهائل من البيانات إلى عدد قليل من المجموعات أو الفئات، وذلك من اجل تسهيل عمليه التحليل.

خوارزميات التجمع تستخدم على نطاق واسع ليس فقط لتنظيم وتصنيف البيانات وإنما هي مفيدة لضغط البيانات وبناء نموذج ترتيب البيانات، حيث انه اذا كان بإمكاننا أن نجد مجموعات من البيانات، فانه بالإمكان بناء نموذج للمشكلة على أساس تلك المجموعات.

هناك عدد من التقنيات المستخدمة في عملية تجميع البيانات، ومن هذه التقنيات (الخوارزميات) التي سوف يتم الحديث عنها بشكل مفصل:

- K-means Clustering
- Subtractive Clustering

1- خوارزمية الـ (K-means Clustering) :

هي خوارزميه لجمع عدد من البيانات استنادا إلى خصائص وسمات هذه البيانات، وتتم عمليه التجميع من خلال تقليل المسافات بين البيانات ومركز التجمع (centered cluster).
وتتم هذه الخوارزمية من خلال الخطوات التالية :

1- حساب إحداثيات مراكز التجميع
2- حساب المسافة بين كل البيانات ومراكز التجميع.
3- تجميع البيانات وتنظيمها في مجموعات بناء على اقل المسافات بين المركز ونقاط البيانات.
4- أعاده تنفيذ الخطوات من 1 – 3 حتى الوصول إلى حاله الثبات

يعتمد أداء هذه الخوارزميه على المواقع الاوليه لمراكز التجمع (Centered)، ومن المستحسن تنفيذ هذه الخوارزميه عدة مرات مع اختلاف المراكز في كل مرة عن المرات السابقة.

نفترض لدينا أربعة أنواع من الأدوية، وكل نوع من الأدوية لديه عدد من السمات، في هذا المثال كل نوع له سمتان.

نوع الدواء(Medicine)	مؤشر الوزن(Weight Index)	معامل الحموضة(PH)
A	1	1
B	2	1
C	4	3
D	5	4

الهدف من هذا المثال هو جمع أنواع هذه الأدوية في مجموعتين اعتمادا على سمات كل نوع من الأدوية، ولتحقيق هذا الهدف علينا تنفيذ خطوات خوارزمية التجميع كالآتي :

1.القيم الابتدائية لمراكز التجمع:

نفترض أن الدواء A والدواء B هما مراكز التجمع الأولى، لتكن c1 و c2 تدل على إحداثيات المراكز، حيث أن c1=(1,1) وc2=(2,1).

يبين الشكل توزيع أنواع الأدوية المعبر عنها بالمعين الأزرق على المستوى الريكاردي، كما يبين مراكز التجمع الابتدائية، مع الأخذ بعين الاعتبار أن هذه المراكز تم افتراضها بشكل عشوائي.

3. المسافات بين النقاط والمراكز:

نحسب المسافة بين مركز التجمع وكل نقطة من النقاط في المستوى فينتج لدينا مصفوفة من المسافات، حيث إن كل عمود في مصفوفة المسافات يمثل نوع دواء واحد، الصف الأول من مصفوفة المسافات يتكون من المسافات بين كل نقطة ومركز التجمع الأول، والصف الثاني يتكون من المسافات بين كل نقطة ومركز التجمع الثاني.

3. تجميع النقاط:

حيث نحيل كل نقطة إلى مركز تجمع بالاعتماد على أقل مسافة، وهكذا فان الدواء الأول (A) ينتسب إلى المجموعة الأولى، الدواء الثاني (B) إلى المجموعة الثانية، الدواء الثالث (C) إلى المجموعة الثانية، والدواء الرابع (D) يعود للمجموعة الثانية.

ينتج لدينا مصفوفة المجموعات G التي تتكون من القيم 1 و 0 ، ويكون العنصر في مصفوفة المجموعات يساوي 1 فقط إذا كان الدواء مسند إلى تلك المجموعة.

4. التكرار الأول، تحديد مراكز التجمع:

بعد معرفة عناصر كل مجموعة، نحسب مركز جديد لكل مجموعة اعتمادا على هذه العضويات الجديدة، المجموعة الأولى تتكون من عنصر واحد فقط وتبقى إحداثيات مركز التجمع الأول كما هي دون تغيير c1=(1,1).

أما المجموعة الثانية والتي تتكون من ثلاث عناصر، تتغير إحداثيات مركز التجمع الثاني بالاعتماد على إحداثيات العناصر الثلاثة.

5- التكرار الأول، المسافات بين النقاط والمراكز:

في هذه الخطوة يتم حساب المسافة بين كل نقطة ومراكز التجمع الجديدة، كما في الخطوة الثانية، ينتج لدينا مصفوفة من المسافات.

6. التكرار الأول، تجميع النقاط:

على غرار الخطوة الثالثة، نحيل كل نقطة إلى مركز تجمع بالاعتماد على أقل مسافة، بالعودة إلى مصفوفة المسافات الجديدة، ننقل الدواء الثاني (B) إلى المجموعة الأولى، بينما تبقى باقي الأدوية كما هي فتظهر مصفوفة المجموعات.

7. التكرار الثاني، تحديد مراكز التجمع:

الآن نقوم بإعادة الخطوة الرابعة لحساب إحداثيات مراكز التجمع الجديدة بالاعتماد على عملية التجميع في التكرار الأول حيث تتكون كل من المجموعة الأولى و الثانية من عنصرين.

8. التكرار الثاني، المسافات بين النقاط والمراكز:

نكرر الخطوة الثانية، فينتج لدينا مصفوفة مسافات جديدة .

9. التكرار الثاني، تجميع النقاط:

مرة أخرى نحيل كل نقطة إلى مركز تجمع بالاعتماد على أقل مسافة.

ينتج لدينا في النهاية بمقارنة التجمع بين التكرار الأول والتكرار الثاني، نلاحظ أن المجموعات لم تتغير من حيث عناصرها وهذا يعني أن عملية الحسابات في الـ (k-mean clustering) وصلت إلى حالة الثبات، وهذا يعني أن هذه الخوارزمية لم تعد بحاجة إلى المزيد من التكرار، وبالتالي حصلنا على النتيجة النهائية للتجميع.

2- **خوارزمية الـ (Subtractive Clustering):**

المشكلة في طريقة التجمع السابقة (Mountain Clustering) هي أن العمليات الحسابية تزداد طرديا بازدياد أبعاد المشكلة، وذلك لأنه وكما ذكر سابقا يتم تقييم الـ (Mountain Function) عند كل نقطة تقاطع في الشبكة على مستوى البيانات.

استطاعت خوارزمية الـ (Subtractive Clustering) حل هذه المشكلة، وذلك بترشيح عدد من نقاط البيانات لتكون مراكز للمجموعات، بدلا من استخدام نقاط تقاطع خطوط الشبكة، كما هو الحال في الـ (MC)، وهذا يعني أن العمليات الحسابية أصبحت تتناسب مع حجم المشكلة بدلا من أبعادها.

خوارزمية الـ (Subtractive clustering) هي عملية تحديد مراكز المجموعات التي تجمعها صفة مشتركة بين كل الأعضاء دون العلم بعدد المجموعات الموجودة لدينا.

وتعتمد هذه الطريقة على حساب كثافة البيانات عند كل نقطة ضمن مستوى معين، فإذا كانت كل نقطة مرشحة لتكون مركز تجمع، فانه يمكن قياس كثافة البيانات عند النقطة x_i من المعادلة التالية:

حيث أن r_a ثابت موجب يمثل قطر دائرة حول كل نقطة، يتم حساب الكثافة داخل هذه الدائرة، وكلما كبر هذا القطر أصبح لدينا عدد اقل من المجموعات، وكلما قل القطر زاد عدد المجموعات، ودائما تكون قيمة r_b أكبر من قيمة r_a (غالبا يستخدم $r_b=1.5r_a$)، وذلك لتقليل قيم الكثافة عند النقاط المجاورة لنقطة المركز الأولى.

تم اختيار المركز الأول xc1 والذي كانت كثافة البيانات عنده أعلى ما يمكن Dc1 ، بعد ذلك يتم حساب قيم الكثافات الجديدة عند كل نقطة xi .

وتقوم خوارزمية الـ (Subtractive clustering) بالخطوات التالية:
1- إيجاد نقطه معينه موجودة في المجال تكون عندها الكثافة عاليه ويتم حساب الكثافة من المعادلة الأولى ومن ثم اختيار نقطه معينه كمركز، وذلك عن طريق وجودها بين عدد كبير من النقاط المجاورة .
2- يتم حذف نقاط البيانات.
3- ثم تبحث الخوارزمية عن مركز جديد، وذلك عن طريق حساب قيم الكثافة للنقاط الأخرى كما في المعادلة الثانية ، وتستمر هذه العملية حتى الانتهاء من كل النقاط أو إيجاد عدد كاف (مناسب) من المجموعات.
أحد أبرز ميزات هذه الخوارزمية، هي أنها أكثر فعالية من الخوارزميات التي ذكرت سابقا، كما أنها الأسرع في تشكيل المجموعات.

4-5 :خوارزمية (Dijkstra):

إيدسجر ديكسترا (Edsger Dijkstra) هو أحد العلماء البارزين في علوم الحاسب. ولد إيدسجر الهولندي الأصل سنة 1930م في مدينة روتردام ، وبدأ مشواره التعليمي بمجال الفيزياء النظرية في جامعة ليدن لكن سرعان ما أدرك أن اهتمامه منصب في علوم الحاسب .

استلم ديكسترا عام 1972م جائزة A. M . Turing نظير مساهمته الأساسية في برمجة اللغات، كما احتفظ بمنصبه في (كرسي سكالمرجر المئوي لعلماء الحاسب) في جامعة تكساس في بوسطن منذ عام 1984م وحتى تقاعد عام 2000م .

من أبرز إسهاماته في علوم الحاسب هي خوارزمية الطريق الأقصر والتي عرفت أيضاً بخوارزمية ديكسترا، استخدمت هذه الخوارزمية في تنظيم نقل المعلومات بين أجهزة الحاسب وعرفت فيما بعد بخوارزمية الطريق الأقصر الأول المفتوح .

كتب ديكسترا عام 1968م ورقتي بحث مهمتين مخصصتين لنظام البرمجة المتعددة وعمليات التعاون التسلسلية، واشتهر أيضاً بابتكاره لعبارة البرمجة الشهيرة (اثنان أو أكثر تستخدم التكرار "2 for a use more r " (والتي تشير إلى حقيقة أنه حينما تجد نفسك تقدم أكثر من مثال لبنية معلوماتية فإنه حان الوقت لتلخص هذا المنطق داخل حلقة تكرارية .

والخوارزمية الخاصة بهذه الطريقة هي :

```
DIJKSTRA (G, w, s)

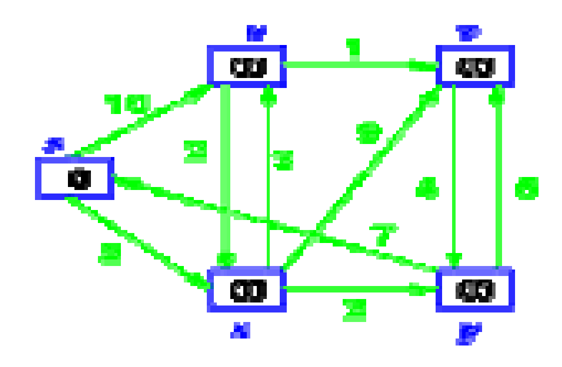
    1.    INITIALIZE SINGLE-SOURCE (G, s)
    2.    S ← { }    // S will ultimately contains vertices of final
                        shortest-path weights from s
    3.    Initialize priority queue Q i.e., Q ← V[G]
    4.    while priority queue Q is not empty do
    5.       u ← EXTRACT_MIN(Q)    // Pull out new vertex
```

```
6.        S ⟵ S{u}
                    // Perform relaxation for each vertex v
                 adjacent to u
7.        for each vertex v in Adj[u] do
8.         Relax (u, v, w)
```

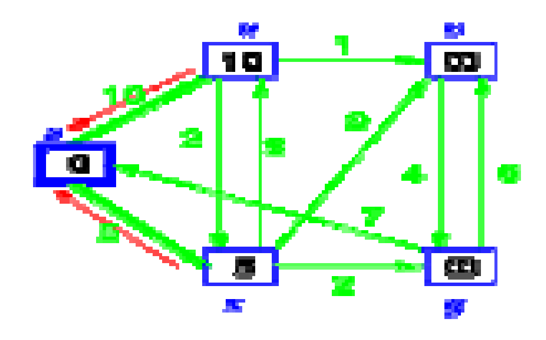
5-5 : أمثلة لتطبيق خوارزمية (Dijstra):

مثال// لتطبيق الخوارزمية كما في الخطوات الآتية:

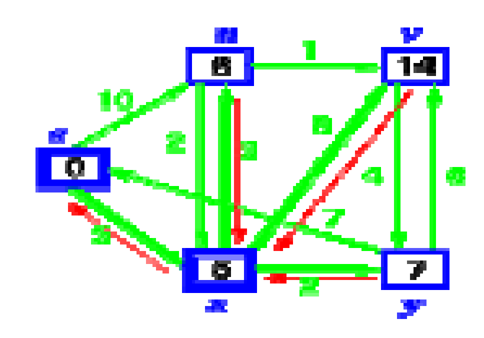
1- البدء بمخطط G=(V, E) كل العقد تمتلك كلف غير منتهية عدا العقدة S حيث كلفتها 0



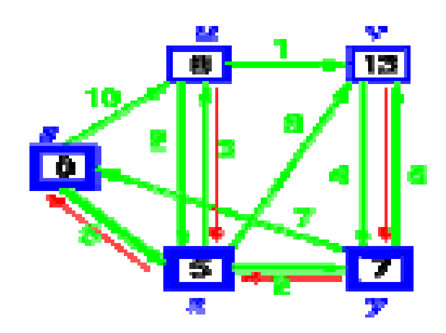
2- أولا نختار عقدة قريبة من S وإيجاد d[s]



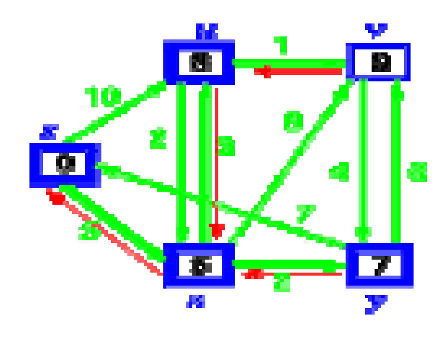
3- اختيار عقدة x وتطبيق خطوات الخوارزمية



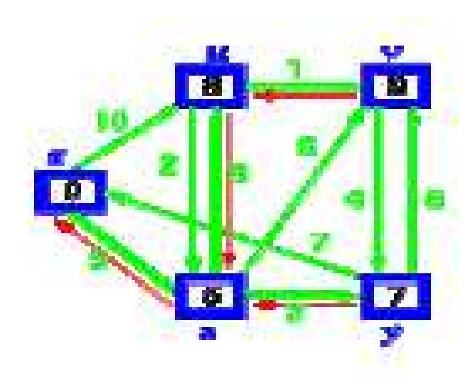
4- اختيار y قريبة للعقدة S وتطبيق بقية الخطوات



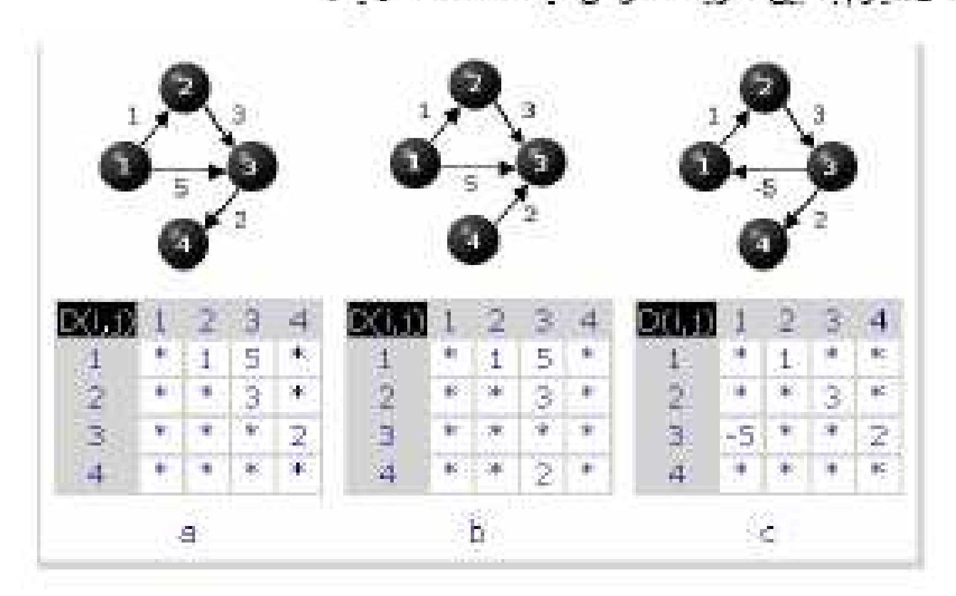
5- ألان نمتلك العقدة u نختار عقدة جارتها V



6- علما إن المخطط سوف يعطي اصغر مسار v اخيرا تضيف العقدة



مثال// يقوم بتطبيق الطريقة ، افترض لديك المخططات الآتية :



D(i,j)	1	2	3	4
1	*	1	5	*
2	*	*	3	*
3	*	*	*	2
4	*	*	*	*

a

D(i,j)	1	2	3	4
1	*	1	5	*
2	*	*	3	*
3	*	*	*	*
4	*	*	2	*

b

D(i,j)	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

c

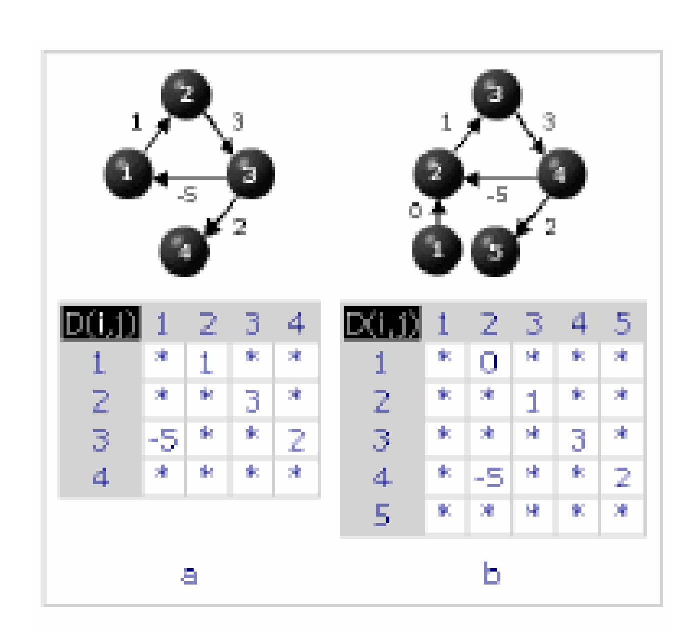
Let C={1,2,...,n} denote the set of cities and for each city j in C let P(j) denote the set of its *immediate predecessors*, and let S(j) denote the set of its *immediate successors*, namely set

P(j) = {k in C: D(k,j) < infinity} , j in C
S(j) = {k in C: D(j,k) < infinity} , j in C

Thus, for the problem depicted in Figure 1(a), P(1)={}, P(2)={1}, P(3)={1,2}, P(4)={3}, S(1)={2,3}, S(2)={3}, S(3)={4}, S(4)={}, where {} denotes the empty set.

Also, let NP denote the set of cities that have no immediate predecessors, and let NS denote the set of cities that have no immediate successors, that is let

```
NP = {j in C: P(j) = {}}
NS = {j in C: S(j) = {}}
```



D(i,j)	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

a

D(i,j)	1	2	3	4	5
1	*	0	*	*	*
2	*	*	1	*	*
3	*	*	*	3	*
4	*	-5	*	*	2
5	*	*	*	*	*

b

Range | Sparsity | Acyclic | D(i,j) < 0 | Gen

Clear | Status: Generated new problem. Now, idle ... | Solve: | Help

	1	2	3	4	5	6	7	8	9	10
NU(j)	?	?	?	?	?	?	?	?	?	?
OIP(j)	?	?	?	?	?	?	?	?	?	?
f(j)	?	?	?	?	?	?	?	?	?	?

D(i,j)	1	2	3	4	5	6	7	8	9	10
1		8	2	6	10	7	8	5	3	2
2			3	5	7	2	10	2	9	3
3		*		7	6	3	8	4	5	7
4		*	*		4	7	8	4	7	7
5		*	*	*		10	5	7	3	5
6		*	*	*	*		2	4	6	8
7		*	*	*	*	*		6	9	7
8		*	*	*	*	*	*		9	7
9		*	*	*	*	*	*	*		8
10		*	*	*	*	*	*	*	*	

يتم هنا إيجاد اقصر المسافات:

x_{ij} = Quantity (flow) sent along the link from city i to city j; i,j=1,2,...,n

Then the minimum cost network flow problem is as follows:

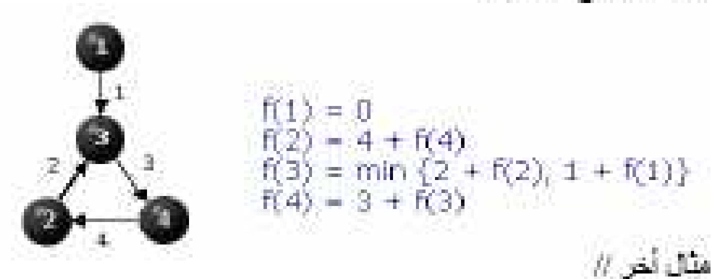
$$\min \Sigma_{ij} c_{ij} x_{ij}$$

s.t.

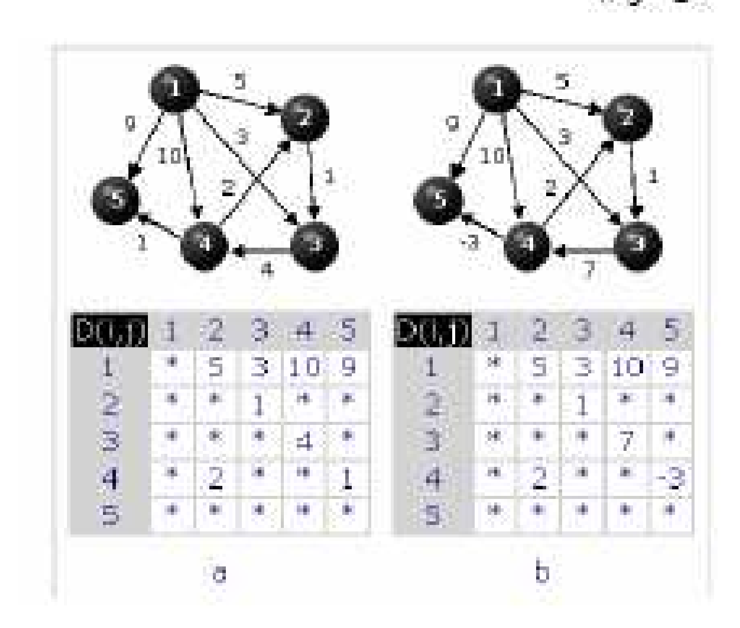
$$\Sigma_k x_{jk} - \Sigma_i x_{ij} = s_j \text{ , } j=1,2,\dots,n$$

$$x_{ij} >= 0 \text{ ; } i,j=1,\dots,n$$

مثال// يوضح الطريقة .



مثال آخر //



D(i,j)	1	2	3	4	5
1	*	5	3	10	9
2	*	*	1	*	*
3	*	*	*	4	*
4	*	2	*	*	1
5	*	*	*	*	*

D(i,j)	1	2	3	4	5
1	*	5	3	10	9
2	*	*	1	*	*
3	*	*	*	7	*
4	*	2	*	*	-3
5	*	*	*	*	*

الحل//

Iteration	k	U	F
0	--	{1,2,3,4,5}	(0,*,*,*,*)
1	1	{2,3,4,5}	(0,5,3,10,9)
2	3	{2,4,5}	(0,5,3,7,9)
3	2	{4,5}	(0,5,3,7,9)
4	4	{5}	(0,5,3,7,8)

a

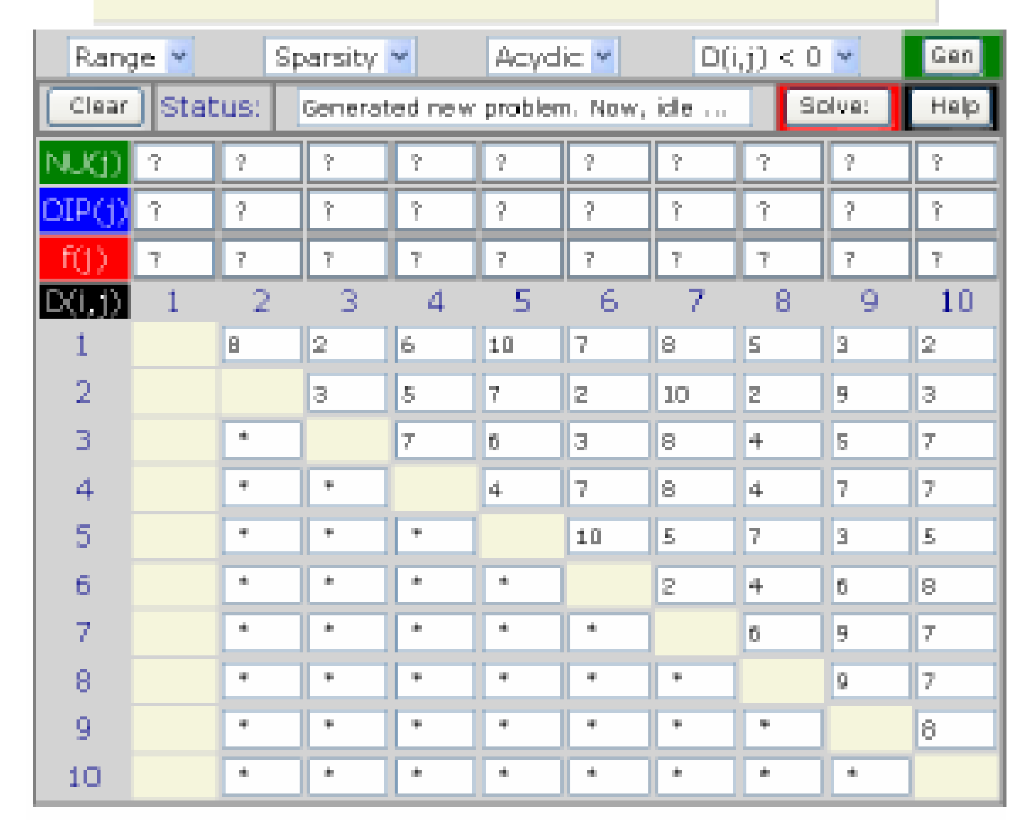
Iteration	k	U	F
0	--	{1,2,3,4,5}	(0,*,*,*,*)
1	1	{2,3,4,5}	(0,5,3,10,9)
2	3	{2,4,5}	(0,5,3,10,9)
3	2	{4,5}	(0,5,3,10,9)

b

$F(n) = F(5) = 9 > f(n) = f(5) = 7.$

الخوارزمية التابعة للمثال تكون كالآتي

```
Initialize: k=1; F(1) = 0; F(j) = Infinity, j=2,...,n
            U = {1,2,3,...,n}
  Iterate: While (|U| > 1 and F(k) < infinity) Do:
              U = U\{k}
              F(j) = min {F(j),D(k,j) + F(k)}, j in U/\S(k)
              k = arg min {F(i): i in U}
           End Do.
```



NU(j)	?	?	?	?	?	?	?	?	?	?
OIP(j)	?	?	?	?	?	?	?	?	?	?
f(j)	?	?	?	?	?	?	?	?	?	?
D(i,j)	1	2	3	4	5	6	7	8	9	10
1		8	2	6	10	7	8	5	3	2
2			3	5	7	2	10	2	9	3
3		*		7	6	3	8	4	5	7
4		*	*		4	7	8	4	7	7
5		*	*	*		10	5	7	3	5
6		*	*	*	*		2	4	6	8
7		*	*	*	*	*		6	9	7
8		*	*	*	*	*	*		9	7
9		*	*	*	*	*	*	*		8
10		*	*	*	*	*	*	*	*	

مثال// استخدام خوارزمية ديكسترا في حركة الربوت وايجاد اقصر مسافة.

Example arena:
X-------
-B----B-
B-------
--B-----
----BB--
----B---
-----B-B
-----B-B
B------Y

Input:
8 8
10
1 1
1 6
2 0
3 2
4 4
4 5
5 4
6 5
6 7
7 0

Sample Output:
0 0
0 1
0 2
1 2
2 2
2 3
2 4
2 5
2 6
3 6
4 6
5 6
6 6
7 6
7 7
1- 1-

PROGRAM: (TRIED AND TESTED IN TURBO C++

```
-------------------------------------------------------
# include <stdio.h>
# include <stdlib.h>
struct Matrix { short int **array;
                int row;int col;
                }
struct Vertex { int num ; int currDist ;
            };
typedef struct Matrix matrix;
typedef struct Vertex vertex;
void getGrid(matrix &m);
void getShortestPath(); /* Dijkstra Algorithm */
void printSolution(int p[],int index);
matrix m;
int main()
{
getGrid(m);
getShortestPath();
printf("\n-1 -1\n\n");
free(m.array);
return 0;
}
void getGrid(matrix &m)
{int ctr1,ctr2,blockedSquares,x,y;
scanf("%d%d%d",&m.row,&m.col,&blockedSquares);
m.array=(short int **)malloc(m.row*sizeof(short int *)) ;
for(ctr1=0;ctr1<m.row;ctr1++)
m.array[ctr1]=(short int *)malloc(m.col*sizeof(short int));
for(ctr1=0;ctr1<m.row;ctr1++)
 for(ctr2=0;ctr2<m.col;ctr2++)
  m.array[ctr1][ctr2]=0;
for(ctr2=0;ctr2<blockedSquares;ctr2++)
 {
  scanf("%d%d",&x,&y);
  m.array[x][y]=1;
 };
}
```

```
void getShortestPath() /* Uses Dijkstra Algorithm */
{
int ctr1=0,ctr2=0,row1,col1,row2,col2;
int *predecessor=(int *)malloc(sizeof(int) * m.row * m.col) ;
vertex *toBeChecked,minVertex;
toBeChecked=(vertex *)malloc(sizeof(vertex) * (m.row*m.col+1));
for(ctr1=1;ctr1<=m.row*m.col;ctr1++)
        {predecessor[ctr1-1]=31000;
        toBeChecked[ctr1].num=ctr1-1;
        toBeChecked[ctr1].currDist=31000;
        };
predecessor[0]=0;
toBeChecked[0].num=toBeChecked[0].currDist=m.row*m.col;
toBeChecked[1].currDist=0;
while(toBeChecked[0].num!=0)
  {
  minVertex=toBeChecked[1];ctr2=1;
  for(ctr1=1;ctr1<=toBeChecked[0].num;ctr1++)
                {
                if(toBeChecked[ctr1].currDist<minVertex.currDist)
                 {ctr2=ctr1;minVertex=toBeChecked[ctr1];
                };
                };
   toBeChecked[ctr2]=toBeChecked[toBeChecked[0].num];
   toBeChecked[0].num--;
    row1=minVertex.num/m.col;col1=minVertex.num%m.col;
        for(ctr1=1;ctr1<=toBeChecked[0].num;ctr1++)
        {
          row2=toBeChecked[ctr1].num / m.col ;
          col2=toBeChecked[ctr1].num % m.col ;
          if(m.array[row2][col2]==0)
          if(((col1-col2)*(col1-col2) == 1 && row1 == row2)||((row1-
                              row2)*(row1-row2)==1 && col1==col2))
        if(toBeChecked[ctr1].currDist>minVertex.currDist+1)
        {
          toBeChecked[ctr1].currDist=minVertex.currDist+1;
           if(toBeChecked[ctr1].num!=0)
        predecessor[toBeChecked[ctr1].num]=minVertex.num;
                        };
                }
   }
```

```
printSolution(predecessor,m.row*m.col-1);
}
void printSolution(int p[],int index)
{
if(index==0)
 { printf("\n0 0");
   return ;
 };
if(p[index]==31000)
  return;
printSolution(p,p[index]);
printf("\n%d %d",index/m.col,index%m.col);
}
```

6-5: المخططات متعددة المراحل (Multistory graph) :

هو مخطط موجه فية تقسم العقد إلى ($K \geq 2$) مجموعه منفصلة (Vi) حيث ($1 \leq i \leq k$). المجموعتان (V_1 , V_k) تتكونتان بحيث ($|V_1| = |V_k| = 1$) (أي انه عدد العناصر الموجودة =1) حيث إن المجموعة الأولى والثانية تحتوي على عقدة واحدة.

افترض إن S هي عقدة البداية في V_1 وان t هي عقدة النهاية في V_k وافترض إن $c[i, j]$ تمثل كلفة الحافة بين (i, j) كما في المعادلة التالية :

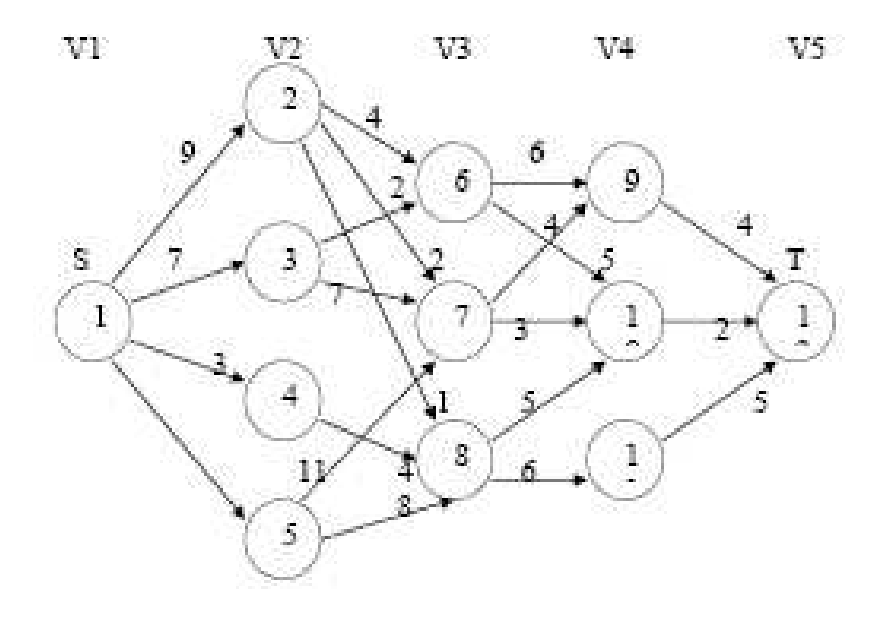
$$Cost(i, j) = \min\{c(j, r) + cost(i+1, r)\}$$

كما يرمز للحافة الموجودة بكل مخطط (<i,j>)

إن كلفة المسار من البداية S إلى النهاية T هي مجموع كلف الحواف على المسار .

ملاحظة // مسألة المخطط متعدد المراحل هي إيجاد مسار اقل كلفة من (S إلى T)

كل مجموعه V_i تمثل مرحلة في المخطط وكل مسار من (S إلى T) يبدأ (بالمرحلة 1 وينتهي بالمرحلة k).



شكل رقم (24) يوضح مخطط متعدد المراحل

صياغة البرمجة الديناميكية لمسالة مخطط V_k من المراحل :

6-5 -1 الطريقة التصاعدية (Forward approach):

نلاحظ إن كل مسار من (S إلى T) يكون نتيجة لتعاقب (k-2) قرار ، حيث كل قرار (i) يشمل تحديد أية عقدة ($V_i + 1$) حيث ($1 \leq i \leq k - 2$) تكون على المسار.

ولو فرضنا إن $P(i, j)$ هو المسار الأقل كلفة من العقدة (j) في المجموعه V_i إلى العقدة (t) بمعنى إن $P(2,2)$ حيث (j) يمثل عقدة و (i) يمثل مرحلة من المراحل وان $Cost\ (i, j)$ يمثل كلفة ذلك المسار بمعنى $Cost\ (2,2)$ وباستعمال الطريقة التصاعدية (أي أنها تحل تناقصياً) معنى ذلك إننا نبدا من الأخير(العقدة T) إلى البداية(العقدة S).

$$Cost\ (i, j) = \min mum\ \{c[j, r] + \cos t(i + 1, r)\} \ldots\ldots .1$$

وهي كلفة المسار الأقل كلفة حيث :

$$r \in V_i + 1, < j, r > \in E$$

أي إن $< r, j >$ هي حافة تنتمي إلى جميع الحواف الموجودة في المخطط وحيث :

$$Cost\ (k - 1, j) = \begin{cases} c[j, t] & if < j, t > \in E \\ \infty & if < j, t > \notin E \end{cases}$$

ولهذا فان المعادلة رقم (1) تحل للحالة $Cost(1,S)$ بحساب أولا :

$$Cost\ (k - 2, j), \forall j \in V_{k-2}$$

ثم بعد ذلك

$$Cost\ (k - 3, j), \forall j \in V_{k-3}$$

وهكذا نستمر إلى إن نصل إلى العقد الموجودة في $V(1)$ واخيراً $Cost\ (1, S)$ حيث يتم خزن قيمة المسار الأقل كلفة .

وباعتبار المخطط الموجود في الشكل رقم (1) السابق نحصل على

$$Cost\ (3,6) = \min\{6 + \cos t(4,9), 5 + \cos t(4,10)\} = 7$$

وهي قيمة المسار الأقل كلفة

$$Cost\ (3,7) = \min\{4 + \cos t(4,9), 3 + \cos t(4,10)\} = 5$$

$$Cost\ (3,8) = \min\{5 + \cos t(4,10), 6 + \cos t(4,11)\} = 7$$

$$Cost\ (2,2) = \min\{4 + \cos t(3,6), 2 + \cos t(3,7), 1 + \cos t(3,8)\} = 7$$

$$Cost\ (2,3) = \min\{2 + \cos t(3,6), 7 + \cos t(3,7)\} = 9$$

$$Cost\ (2,4) = \min\{11 + \cos t(3,8)\} = 18$$

$$Cost\ (2,5) = \min\{11 + \cos t(3,7), 8 + \cos t(3,8)\} = 15$$

$$Cost\ (1,1) = \min\{9 + \cos t(2,2), 7 + \cos t(2,3), 3 + \cos t(2,4), 2 + \cos t(2,5)\} = 16$$

لاحظ إننا نأخذ الكلف الأقل بصورة متسلسلة دائماً لانه الأفضل ,حيث إن أخر كلفة مسار تم حسابها هي كلفة $Cost\ (1,1)$ وتعني كلفة $Cost\ (1, S)$.

ولهذا فان المسار الأقل كلفة من (S إلى T) كان بكلفة مقدارها 16 ولتحديد المسار تسجل القرارات المتخذة في كل حالة (عقدة) .
ولنفترض إن $D[i, j]$ هي قيمة r التي تصغر العلاقة التي ذكرناها سابقاً وهي :
$\{ c[j, r] + Cost(i+1, r)\}$
ملاحظة// إن $D[i, j]$ يمثل القرار المتخذ للمرحلة القادمة حيث يكون للعقدة التي تعطي اقل قيمة حسب العلاقة السابقة .
ملاحظة// إن قيم القرارات المتخذة هي نفسها ارقام العقد الموجودة على المسار ودائماً نبدأ بالمرحلة (k-2).

$D[2,2] = 7, D[2,3] = 6$

$D[2,4] = 8$

$D[2,5] = 8$

$D[1,1] = 2$

وباعتبار المسار الأقل كلفة هو :

$S = 1 \quad ,V_2 \quad V_3 .... \quad .... \quad V_{k-1} \quad T = 12$

حيث :
S : تمثل العقدة الأولى وهي الأقل دائماً
2 : $V(2)$
7 : $V(3)$
10 : $V(k-1)$
12 : T وتمثل العقدة الأخيرة

$V_2 = D[1,1] = 2$

$V_3 = D[2, D[1,1]] = D[2,2] = 7$

$V_4 = D[2, D[1,1]] = D[3,7] = 10$

ملاحظة // بعد تحديد العقدة الأولى يتبادر إلى ذهننا السؤال التالي ما هي العقدة الأقل التي يجب إن نختارها في المرحلة التالية .
نستنتج إن المسار الأفضل هو : 1(S) ,2,7,10,12 (T)

وألان سوف نقوم بكتابة الخوارزمية لحل هذه المسالة (خوارزمية المخطط متعدد المراحل المتناظرة للطريقة التصاعدية)
تفترض خوارزمية المخطط متعدد المراحل المتناظرة للطريقة التصاعدية إن العقد V مرتبة من 1 إلى n وان عقدة البداية S تعطي الفهرس 1 ثم تعطي عقد المجموعة V_2 فهارس متتالية حتى تصل إلى عقدة النهاية T ، أي إن الفهارس المعطاة لعقد المجموعة $V_i + 1$ تكون اكبر من تلك المعطاة للـ $V[i]$.

```
Void FGraph (Graph G ,int k,int n ,int P[])
{ float cost [maxsize]
  int D[maxsize] ,r ;
  cost[n]= 0.0;
  for (int j= n-1;j>=1;j--)
  {  // Compute cost[j].
   Let   r  be Vertex Suchthat <j,r> is an edge ,of G and c[j][i]+cost[r]
     Is minmum ;
   Cost[j]= c[j][r]+ cost[r] ;
   D[j]= r ;
  }
  P[1]=1;  p[k]=n;
  For(j=2; j<= k-1; j++)
         P[j] =D[ p[j-1]];
}
```

ولتوضيح خطوات تنفيذ الخوارزمية يتم كالتالي :

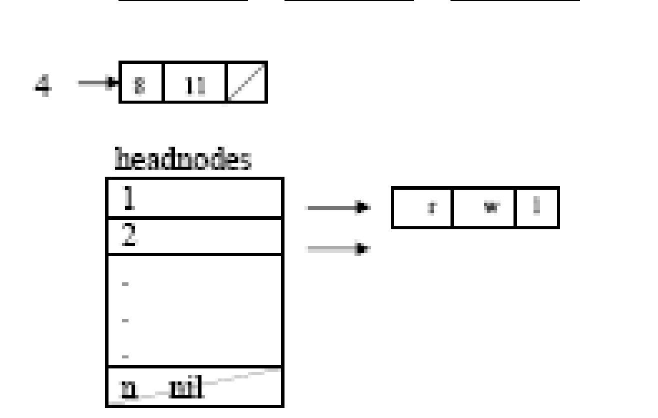
- إن المصفوفة [ ]P تحوي ارقام العقد الموجودة على المسار ، ولحساب الكلف احتجنا مصفوفة أحادية وليس ثنائية وهي مصفوفة [ ]cost ، أما المتغير r فهو يمثل العقد الموجودة في المرحلة الثانية .
- إن أول عقدة نبدا بحساب الكلفة لها هي العقدة الاخيرة لذلك خصصنا cost[n]= 0.0 .
- لإيجاد قيمة r يتم بالاعتماد على إن (كلفة r مضافاً اليها قيمة العلاقة) يجب إن تكون اقل ما يمكن.
- ولبناء المخطط Graph وقراءته بلغة ++C يتم كالتالي :

المخطط يتكون من مصفوفة مؤشرات إلى قيود (تدعى طريقة قوائم التجاور) كما يلي :

```
Struct  node
{  int rertex ;
   float weight ;
   Struct  node*link ;
}
```

- إن العقدة تمثل بالمتغير rertex ، اما القيمة الموجودة على الحافة فهي تمثل بالمتغير weight وطبعا هذا الجزء من البرنامج تم لعقدة واحد فقط .
- اما بالنسبة لمجموعة عقد فيتم كالتالي :

```
Struct node*headnods[n] ;
```



- لاحظ ان العقدة الاولى ترتبط باربعة عقد اخرى لذلك استخدمنا headnods
- ان عملية ادخال وقراءة البيانات وتمثيل المخطط يجب ان تتم قبل تنفيذ الخوارزمية FGraph ومن ثم يتم استدعائها.

وهذا هو جزء البرنامج الخاص بهذة الطريقة التصاعدية (Forward approach) :

```
FGraph  ( Type head*node )
{ Type   *p=newType;
  q=p;
   p->weight;
   p->rertex;
   for(int i=2;i<=m;i++)
      {  type *p;
         p->weight;
```

```
        p->rertex;
        q->link=p;
      }
  q->link=0;
}
head[n]=0;

void FGraph (Type *head[20],int k,int n)
{ float cost [100],x; int p[100],D[100];
   cost[n]=0;
  for(int j=n-1;j>=1;j--)
   {cost[j]=head[j]->weight+cost[head[j]->rertex];
    D[j]=head[j]->rertex;
    Head[j]=head[j]->link;
    While (head[j] != 0)
     { x=head[j]->weight +cost[head[j]->rertex];
       If  (x< cost[j])
          { cost[j]=x;
            D[j]=head[j]->rertex;
          }
       head[j]=head[j]->link;
     }
  }
}
P[1]=1;  p[k]=n;
for(int j=2;j<=k-1;j++)
   P[j]= D[p[j-1]];
for(int i=1;i<=k;i++)
   cout<< p[i] << "     "  ;
}
```

تحليل تعقيدات الدالة **(FGraph)**:

في حالة تمثيل المخطط بقوائم التجاور(Linked list) فان r يمكن ايجادها في وقت يتناسب مع درجة العقدة (j) (عدد ارتباطات العقدة مع العقد التي بعدها) (الاسهم) حيث (j) هي من 1 إلى n فإذا كان المخطط $|E|$ من الحواف فان وقت دوارة for الأولى هو:

$$\Theta(|V| + |E|)$$

حيث V تمثل عدد مجموعه العقد ، أما E فهي تمثل عدد الحواف.

أما بالنسبة لوقت دوارة for الثانية فهو $\Theta(k)$ حيث k يمثل عدد المراحل .
هذا يعني إن التعقيدات الكلية للخوارزمية هي

$$\Theta\ (\ |V|\ +\ |E|\ )$$

بالإضافة إلى الخزن الذي تتطلبه المدخلات توجد حاجة لوجود خزن للكلف في المصفوفتين P,D .

2-6-5 :الطريقة التناقصية (Backward approach) :

إن تحديد العقد هنا يكون من النهاية إلى البداية أي من (T إلى S) متحل هذه الطريقة حلاً تصاعدياً وعلاقات حساب الكلفة تتم كالآتي:

$$bCost\ (i,\ j)\ =\ \min\{\ bc\ (i-1,\ r)\ +\ c[r,\ j]\} \ldots\ldots\ 1$$

حيث $r \in V_{i-1}$ و $< r, j > \in E$

إن كل العقد الموجودة في المرحلة الثانية $b\cos t(2,j)$ هي نفسها كلفة العقدة التي تربط العقدة الأولى بالعقدة الثانية

$$b\cos\ t(2,\ j)\ =\ \begin{cases} c[1,j] & if\ <1,j> \in E \\ \infty & if\ <1,j> \notin E \end{cases}$$

توضيح : إذا كانت $bp\ (i,\ j)$ هو مسار اقل كلفة من العقدة S إلى العقدة (j) في المجموعه b_i وكانت $b\cos t(i,\ j)$ هي كلفة المسار $bp\ (i,\ j)$ بوجود العلاقات أعلاه فانه يمكن حساب $b\cos t(i,j)$ وذلك بحساب $b\cos t(i,j)$ للقيمة i=3 و j=4 وهكذا بالنسبة لبقية العقد .

للمخطط السابق المرسوم في الشكل رقم (24) فانه لحساب الكلفة فان أول مرحلة نحسبها هي الثالثة ,وهنا ينبغي علينا ملاحظة الأسهم الداخلة للعقد .

$$b\cos\ t(3,6) = \min\{\ b\cos\ t(2,2) + 4, b\cos\ t(2,3) + 2\} = 9$$

تنوية : للسيطرة على تتبع الحل نضع خطاً اسفل العقدة التي تعطي القيمة لأنها سوف تفيدنا في حساب أو اتخاذ القرارات.
ونطبق بنفس الطريقة بالنسبة لبقية المسارات أي :

$$b\cos\ t(3,7) = 11$$
$$b\cos\ t(3,8) = 10$$
$$b\cos\ t(4,9) = 15$$
$$b\cos\ t(4,10) = 14$$
$$b\cos\ t(4,11) = 16$$

$$\delta \cos t(5,12) = 16$$

أما بالنسبة لإتخاذ القرار فإن المسار الأقل كلفة من (T إلى S) كان بكلفة مقدارها 16
ولتحديد المسار نسجل القرارات المتخذة في كل حالة (عقدة) كما يلي :

$$D[3,6] = 3,\ D[3,7] = 2,\ D[3,8] = 2$$
$$D[4,9] = 6,\ D[4,10] = 7,\ D[4,11] = 8$$
$$D[5,12] = 10$$

$$P[1] = 1$$
$$P[4] = 10$$
$$P[3] = 7$$
$$P[2] = 2$$
$$P[n] = 12$$

وباعتبار المسار الأقل كلفة هو :

$$S = 1 \quad ,V_2 \quad V_3 \ldots\ldots\ldots \quad \ldots. \quad V_{k-1} \quad T = 12$$

حيث :
S : تمثل العقدة الأولى وهي الأقل دائماً
$V(2)$: 2
$V(3)$: 7
$V(k-1)$: 10
T : 12 وتمثل العقدة الاخيرة

$$V_2 = D[3, D[4,10]] = D[3,7] = 2$$
$$V_3 = D[4, D[5,12]] = D[4,10] = 7$$
$$V_4 = D[5,12] = 10$$

نستنتج إن المسار الأفضل هو : 1(S) ,2,7,10,12 (T)
وهذه هي الخوارزمية الخاصة بالمخطط متعدد المراحل المناظرة للطريقة التناقصية (BGraph)

```
Void BGraph (Graph G ,int k,int n ,int P[])
{ float cost [maxsize]
  int D[maxsize] ,r ;
  bcost[1]= 0.0;
  for (int j= 2;j<=n ;j++)
  {  // Compute bcost[j].
   Let   r  be Vertex Suchthat <r,j> is an edge ,of G and bcost[r]+c[r][j]
     Is minmum ;
   bcost[j]= bcost[r] + c[r][j] ;
   D[j]= r ;
  }
  P[1]=1;  p[k]=n;
  For(j= k-1; j>=2; j--)
         P[j] =D[ p[j+1]];
}
```

وهذا هو جزء البرنامج الخاص بهذه الطريقة التناقصية:

```
FGraph  ( Type head*node )
{ Type   *p=newType;
  q=p;
   p->weight;
   p->rertex;
   for(int i=2;i<=m;i++)
      {  type *p;
         p->weight;
         p->rertex;
         q->link=p;
      }
 q->link=0;
}
head[n]=0;

void BGraph (Type *head[20],int k,int n)
{ float bcost [100],x; int p[100],D[100];
  cost[n]=0;
 for(int j=2;j<=n;j++)
  {bcost[j]=head[j]->weight+bcost[head[j]->rertex];
    D[j]=head[j]->rertex;
    head[j]=head[j]->link;
```

```
While (head[j] != 0)
     { x=head[j]->weight +bcost[head[j]->rertex];
       If  (x< bcost[j])
          { bcost[j]=x;
            D[j]=head[j]->rertex;
          }

       head[j]=head[j]->link;
     }
  }
}
P[1]=1;  p[k]=n;
for(int j=k-1;j>=2;j++)
   P[j]= D[p[j+1]];
for(int i=1;i<=k;i++)
   cout<<p[i] <<"      ";
}
```

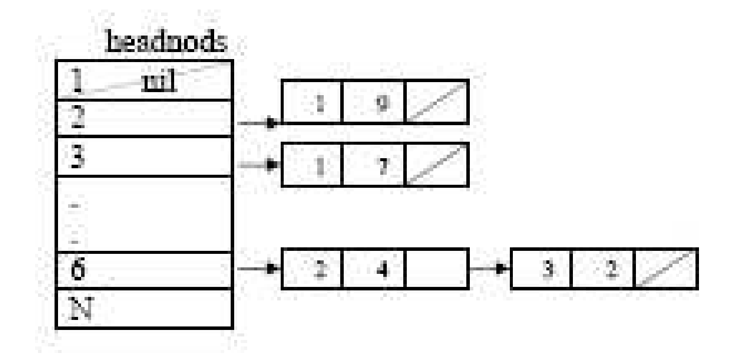
تحليل تعقيدات الدالة (BGraph):

إن تعقيدات الوقت والخزن لهذه الخوارزمية هي نفسها تلك التعقيدات لخوارزمية (FGraph) ولكن بشرط واحد هو تمثيل المخطط بقوائم التجاور معكوسة .

هذا يعني إن التعقيدات الكلية للخوارزمية هي

$$\Theta\ (|V| + |E|)$$

إن قوائم الجوار المعكوسه تمثل كالتالي :



أي إن لكل عقدة P فإنها تملك مجموعه أو قائمة عقد W بحيث $< W , V > \in E$

E تمثل مجموعه الحواف ، V تمثل العقد في قائمة اليسار.

ملاحظة// هناك تطبيقات أخرى للمخططات متعددة المراحل منها ما يخص الموارد (مبلغ من المال يقسم على مجموعه مشاريع) .

3-6-5: طريقة اقتفاء الأثر رجوعاً (Back Tracking) :

تعتبر إحدى أكثر الطرق المهمة لتصميم الخوارزميات حيث تعطي أكثر من حل للمسألة (مجموعه حلول) .
ملاحظة // لغة prolog تستخدم هذا المبدأ في عملها بحيث تبنى كشجرة

تستعمل بمعظم المسائل التي تبحث عن مجموعة حلول أو حل امثل يحقق بعض القيود يكون للحل الصيغة ($X_1, X_2, X_3, ... X_n$) حيث تختار X_i في مجموعة محددة هي S_i
ميزة رئيسية لهذه الطريقة هي انه إذا أدركت إن المتجه الجزئي ($X_1, X_2, X_3, ... X_i$) لن يقود إلى حل امثل فان كل محاولات تكوين المتجهات الجزئية تهمل تماماً .

تتطلب العديد من المسائل التي يتم حلها باستخدام طريقة اقتفاء الأثر رجوعا (إن تحقق الحلول مجموعه من القيود) التي يمكن تقسيمها إلى فئتين وهناك مجموعه من القيود.
المجموعه الأولى صريحه (Explicit Constraints) :
وهي قواعد لتقييم X_i لمجموعه معطاة هي S_i حيث كل المتجهات التي تحقق القيود الصحيحة تكون فراغ الحالات الممكنة .

المجموعه الثانية ضمنية (Implicit Constraints) :
هي قواعد لتحديد أي متجهات الحلول تحقق دالة الهدف أي أنها تصف ارتباطات X_i بغيرها .

الخطوات المشمولة بطريقة اقتفاء الأثر رجوعاً :

1. تعريف فراغ الحلول الممكنة المتضمن الإجابة أو الحل لمثال المسألة
2. تنظيم هذا الفراغ بطريقة مناسبة للبحث (شجرة أو مخطط)
3. بحث الفراغ بطريقة العمق أولا (Depth_first_Search) مع استعمال دوال تقييد (Bounding Function) لتجنب التحرك إلى فراغات جزئية لا تقود إلى حل .

مثال// سنأخذ مسألة n ملكة (n_queens problem) كمثال لتطبيق هذه الطريقة ، حيث توضع الملكات على لوحة شطرنج بشرط إن لا تكون هناك ملكتان على نفس الصف أو القطر أو العمود .
ملاحظة// في هذه المسألة يوجد لدينا n من الملكات يراد وضعها على لوحة شطرنج إبعادها n*n بحيث لا توضع اكثر من ملكة على نفس الصف أو العمود أو القطر .

لنفترض ترقيم صفوف وأعمدة لوحة الشطرنج [n....1] والملكات أيضا [n...1] ،يمكن وضع الملكة i في الصف i ، سنمثل كل حلول مسالة n ملكة بالمتجهات $X_1, X_2, X_3, ... X_n$ حيث X_i هو العمود الذي توضع علية الملكة i بمعنى إذا $X_1 = 2$ معناها الملكة 1 توضع على العمود 2 .

القيود الصريحة هنا هي $S_i = [1 ... n]$

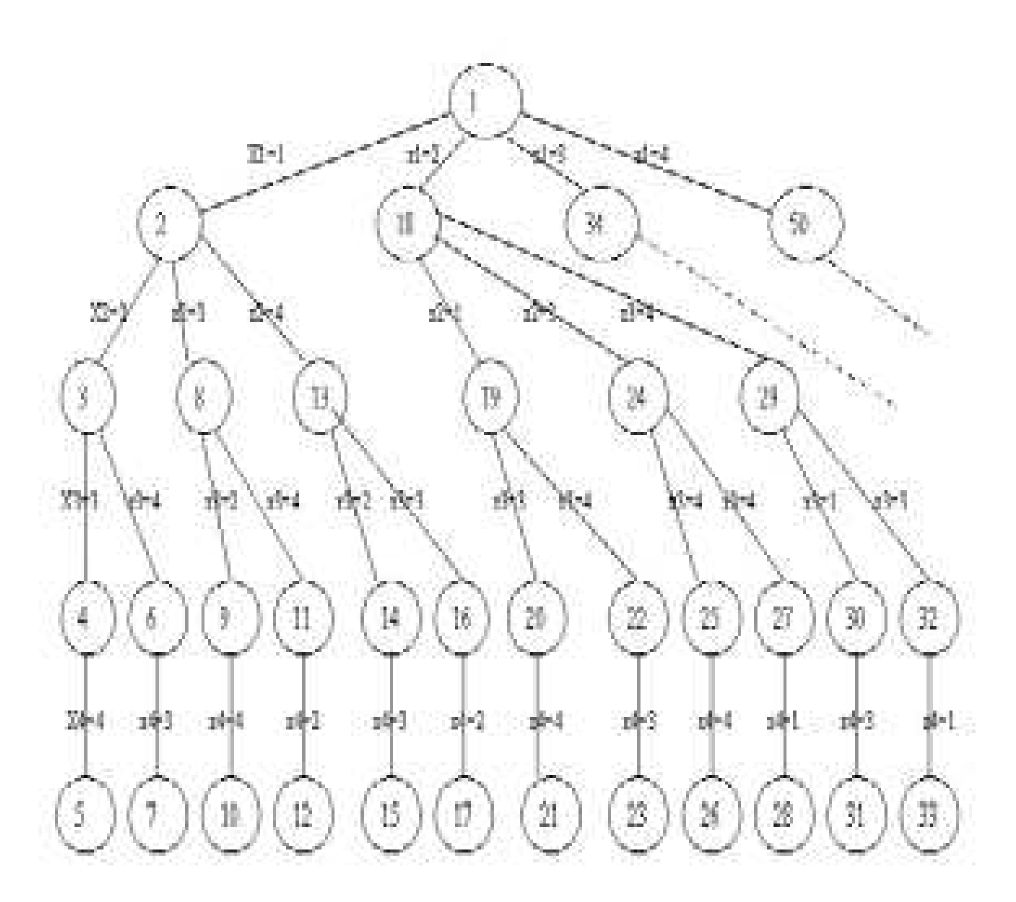
والقيود الضمنية توصف ارتباطات X_i بغيرها

القيود الضمنية هي إن كل الملكات يجب إن تكون على أعمدة وأقطار مختلفة .

القيود الصريحة تعطي متجهات عددها 2^n ، إن القيد الضمني الأول يشير إلى تباديل من المتجهات وهذا يختزل حجم فراغ الحالات أو الحلول من n^n إلى $n!$.

ألان لدينا n=4 وعلينا إيجاد الحلول الممكنة ؟

الحل// بما إن الحجم هو 4 لذا يمكن رسم شجرة الاحتمالات (التباديل) كالتالي :



في هذه المسالة يجب إن نراعي الشرط الذي يقول بان الملكة ممكن إن توضع في العمود الأول أو الثاني أو الثالث أو الرابع كبداية للحل لان المصفوفة فارغة .

بما إن مجموعه الحلول هي !n هذا يعني انه لدينا (24) حلاً وان جزء منها يحقق الحل المطلوب .
الحواف مرقمة من لكل القيم الممكنة X_i ،
الحواف من المستوى i إلى المستوى i+1 تحدد قيم X_i ،لهذا فان الشجرة على أقصى اليسار تحتوي على كل الحلول ذات X_1 تساوي (1).

إن المسار المتولد من العقد التالية (1,18,29,32,33) يعطي حلاً ممكناً يحقق الشرط ، وهو يقابل (2,4,1,3) كما في المصفوفة التالية :

	1	2	3	4
1		X1		
2				X2
3	X3			
4			X4	

هنا x1 نقصد بها الملكة الأولى وهكذا بالنسبة لبقية الملكات .
أي إن:
الملكة 1 توضع بالعمود الثاني
الملكة 2 توضع بالعمود الرابع
الملكة 3 توضع بالعمود الأول
الملكة 4 توضع بالعمود الثالث

تمرين // اكتب برنامج يقوم بتطبيق الخوارزمية الخاصة بمسألة (n_queens problem) ؟

References:

1-Aho, Alfred V. and Jeffrey D. Ullman [1983]. Data Structures and Algorithms. Addison-Wesley, Reading, Massachusetts.

2-Beiler J.:An Introduction to Data Structures ; Allyn and Beacon,Inc,1982.

3-Berman A.M.: Data Structures via C++ (objects by Evolutoin); Oxford University press Inc. ,1997.

4-Berztiss A.T. : Data Structures ,theory and practice ; Academic press Inc ,1975.

5-Cormen, Thomas H., Charles E. Leiserson and Ronald L. Rivest [1990]. Introduction toAlgorithms. McGraw-Hill, New York.

6-Dahl O. J. ,Dijkstra E. W and Hoare C.A.R : Structured 6-programming;Academic press Inc.1972.

7-Dale N. and Lilly S.C: pascal plus Data Structures ,Algorethims and Advance programming ,D.C.Heath and company ,1985.

8-Goodrich M.T. and Tamassia R. : Data Structures and algorithms in java ;John Wiely and Son Inc. ,1998.

9-Gonnet G.H and Baeza –Yates R.:Handbook of Algorethms and data Structures ; Addison_Wesley,1991.

10-Horowitz E.and Sahni S.: Fundamentals of data Structures in pascal ;Computer Science press Inc,1987.

11-Knuth, Donald. E. [1998]. The Art of Computer Programming, Volume 3, Sorting andSearching. Addison-Wesley, Reading, Massachusetts.

12-Pearson, Peter K. [1990]. Fast Hashing of Variable-Length Text Strings.

13-Communications of the ACM, 33(6):677-680, June 1990.

14-Pugh, William [1990]. Skip lists: A Probabilistic Alternative To Balanced Trees.

15-Communications of the ACM, 33(6):668-676, June 1990.

16-Stephens, Rod [1998]. Ready-to-Run Visual Basic Algorithms. John Wiley & Sons,Inc., New York.

17-Thomas H. Cormen ،Charles E. Leiserson ،Ronald L. Rivest ،and Clifford Stein .Introduction to Algorithms ،Second Edition.

18-MIT Press and McGraw-Hill, 2001 .ISBN7-03293-262-0 .Chapter 1: Foundations, pp.3–122 .

19-C.L. PHILIPS& H.T.NAGLE, Digital Control System: Analysis and Design (Printice- Hall 1984).

20-http://i136.photobucket.com/albums/q175/uramium/tab1,2,3.jpg,
21-http://alyaseer.net/files/file.php?id=10
22-http://akoaal.hajznet.com/bubblesortagorith.pdf
23 -http://akoaal.hajznet.com/heapsortagorith.pdf
24-http://akoaal.hajznet.com/meregesortagorith.pdf